भारतीय प्रौद्योगिकी संस्थान रुड़की
Indian Institute of Technology Roorkee

MAJOR PROJECT (BMN610)

ALGORITHMIC TRADING

Submitted to:

Professor Manu Gupta

Department of Management Studies,

Indian Institute of Technology (IIT) Roorkee

Prepared By:

Sachin Sharma

21810058

Date of Submission:

May 01st, 2023

ABSTRACT

Algorithmic trading is a computer-based approach to trading that uses pre-programmed rules and instructions to analyze market data, identify trading opportunities, and automatically execute trades with little to no human intervention. This project focuses on developing and testing a trading algorithm that aims to generate consistent profits using a straddle selling strategy.

The straddle selling strategy involves selling both a call and a put option on the same underlying asset with the same expiration date and strike price. The algorithm developed in this project uses this strategy to identify the best opportunities to execute trades based on pre-programmed rules and instructions. The algorithm was developed using the Python programming language and utilizes broker APIs and data from a WebSocket feed to execute trades in real time.

The primary objective of the algorithm developed in this project is to consistently outperform the general market returns by at least two times by generating profits of 3-5% per month under standard volatility conditions and decent market premiums. The algorithm was tested and tweaked in March to optimize its performance under current low vix conditions, and the final forward testing of the strategy was carried out for the entire month of April using BankNifty derivatives.

The results of the forward testing demonstrated that the performance targets were achieved, and the algorithm has high growth prospects in future years to come. Risk management techniques such as stop-loss orders were implemented to ensure the safety of the trading algorithm. Additionally, live logs and telegram updates were integrated into the algorithm to provide real-time updates from anywhere, enabling traders to monitor the trading activity remotely.

In conclusion, this project demonstrates the potential of algorithmic trading to generate consistent profits with minimal human intervention. The straddle selling strategy, when combined with algorithmic trading, has the potential to outperform the general market returns and generate significant profits. While algorithmic trading has its limitations and risks, careful development and testing of trading algorithms can minimize the risk of unexpected losses.

CONTENTS

Introduction

In today's fast-paced and ever-changing financial markets, traders and investors are constantly looking for ways to gain an edge over their competitors. One such way is through the use of algorithmic trading, commonly referred to as Algo Trading. Algo trading involves the use of computer programs to execute trades automatically based on predefined rules and conditions.

Algo trading has become an integral part of financial markets, and it is used by both individual traders and institutional investors. It is estimated that more than 50% of all trades in the US equity markets are executed through algorithmic trading. This technology has revolutionized the way financial markets operate, providing traders and investors with the tools they need to navigate the complex and volatile markets of today.

The use of Algo Trading has been steadily increasing over the years, and it is now considered to be one of the most popular methods of trading in financial markets. This is due to the many benefits it offers, including speed and efficiency, consistency, backtesting, scalability, and cost-effectiveness.

One of the key advantages of Algo Trading is its speed and efficiency. Algo trading enables traders to execute trades at lightning-fast speeds, allowing them to take advantage of market movements quickly. This helps to reduce the risk of slippage, which occurs when prices change between the time a trade is placed and executed. The speed and efficiency of Algo Trading are essential in today's markets, where a delay of even a few seconds can mean the difference between profit and loss.

Consistency is another key benefit of Algo Trading. Algo trading takes the emotion out of trading, ensuring that trades are executed based on predefined rules and conditions. This helps to eliminate the human error that can occur with manual trading. By removing the emotional aspect of trading, Algo Trading helps traders to stick to their strategies and make more consistent decisions.

Backtesting is another important benefit of Algo Trading. Algo trading allows traders to test their strategies using historical market data to determine their effectiveness. This helps traders to refine their strategies and optimize their trading systems. By backtesting their strategies, traders can identify potential flaws and make improvements before deploying them in real-time trading.

Scalability is another important advantage of Algo Trading. Algo trading allows traders to execute trades across multiple markets and instruments simultaneously, making it easier to diversify their portfolios and manage risk. This is especially important for institutional investors, who need to manage large portfolios across multiple markets.

Finally, Algo Trading can be more cost-effective than traditional trading methods, as it eliminates the need for human traders and reduces the cost of executing trades. This can be especially beneficial for individual traders who may not have the resources to hire a team of traders to execute trades on their behalf.

In conclusion, Algo Trading has become an essential tool for traders and investors in today's financial markets. Its speed and efficiency, consistency, backtesting, scalability, and cost-effectiveness have made it a preferred method of trading for many market participants. As technology continues to evolve, we can expect to see even more sophisticated algorithmic trading systems in the future, further increasing the speed and efficiency of trading.

History, need and Benefits of Algo Trading.

Algo Trading, also known as algorithmic trading, has a long and fascinating history that spans several decades. It has evolved from its early beginnings as a tool for program trading in the 1970s to its current state as a dominant force in financial markets worldwide.

In the early days of Algo Trading, computers were used to help trading desks make informed decisions. However, these early systems were relatively simple and limited in their capabilities. They were primarily used for program trading, which involved the simultaneous buying and selling of a large basket of stocks to take advantage of price discrepancies between them.

In the 1990s, the rise of electronic trading platforms and the availability of real-time market data led to a significant increase in the use of Algo Trading. These advances allowed traders to develop more complex trading algorithms that could analyze vast amounts of data in real time and execute trades automatically based on predefined rules and conditions.

In the early 2000s, the development of high-frequency trading (HFT) algorithms took Algo Trading to a new level. HFT is a form of algorithmic trading that involves the use of complex algorithms to execute trades in microseconds. These algorithms use a variety of techniques, such as statistical arbitrage, to identify and exploit small price discrepancies in the markets.

The use of HFT has grown rapidly in recent years, and it is now estimated that over 70% of all trades in US equity markets are executed through algorithmic trading, with a significant portion of those trades being executed by HFT algorithms. HFT has been the subject of much controversy and debate, with some critics arguing that it has created an uneven playing field for retail investors and small traders.

Despite the controversy, Algo Trading continues to evolve and innovate, with new technologies and techniques being developed all the time. Today, Algo Trading is used by a wide range of market participants, from individual traders to large institutional investors, and it has become an essential tool for navigating the complex and volatile financial markets of today.

One of the key advantages of Algo Trading is its speed and efficiency. Algo Trading enables traders to execute trades at lightning-fast speeds, allowing them to take advantage of market movements quickly. This helps to reduce the risk of slippage, which occurs when prices change between the time a trade is placed and executed. The speed and efficiency of Algo Trading are essential in today's markets, where a delay of even a few seconds can mean the difference between profit and loss.

Consistency is another key benefit of Algo Trading. Algo Trading takes the emotion out of trading, ensuring that trades are executed based on predefined rules and conditions. This helps to eliminate the human error that can occur with manual trading. By removing the emotional aspect of trading, Algo Trading helps traders to stick to their strategies and make more consistent decisions.

Backtesting is another important benefit of Algo Trading. Algo Trading allows traders to test their strategies using historical market data to determine their effectiveness. This helps traders to refine their strategies and optimize their trading systems. By backtesting their strategies, traders can identify potential flaws and make improvements before deploying them in real-time trading.

Scalability is another important advantage of Algo Trading. Algo Trading allows traders to execute trades across multiple markets and instruments simultaneously, making it easier to diversify their portfolios and manage risk. This is especially important for institutional investors, who need to manage large portfolios across multiple markets.

Finally, Algo Trading can be more cost-effective than traditional trading methods, as it eliminates the need for human traders and reduces the cost of executing trades. This can be especially beneficial for individual traders who may not have the resources to hire a team of traders to execute trades on their behalf.

Straddle

The straddle options strategy is a popular strategy used by investors and traders to take advantage of volatile market conditions. It involves buying a call option and a put option on the same underlying asset with the same strike price and expiration date. The investor profits if the price of the underlying asset moves significantly in either direction.

For example, let's say that an investor buys a straddle on Reliance Industries Ltd. The stock is currently trading at Rs. 2,000 per share, and the investor buys a call option with a strike price of Rs. 2,000 and a put option with a strike price of Rs. 2,000. Both options expire in one month.

If the price of Reliance Industries stock goes up to Rs. 2,200, the investor can exercise the call option and buy the stock at Rs. 2,000, then immediately sell it at the current market price of Rs. 2,200, resulting in a profit of Rs. 200 per share. Alternatively, if the price of Reliance Industries stock goes down to Rs. 1,800, the investor can exercise the put option and sell the stock at Rs. 2,000, then immediately buy it at the market price of Rs. 1,800, resulting in a profit of Rs. 200 per share.

The maximum potential loss for a straddle is the total premium paid for the call and put options. This occurs if the price of the underlying asset does not move significantly in either direction before the options expire.

Selling Straddles:

Option selling straddles involves selling both a call option and a put option on the same underlying asset with the same strike price and expiration date. The seller of the Straddle collects a premium from the buyer and profits if the price of the underlying asset remains relatively stable.

For example, let's say that an investor sells a straddle on HDFC Bank Ltd. The stock is currently trading at Rs. 1,500 per share, and the investor sells a call option with a strike price of Rs. 1,500 and a put option with a strike price of Rs. 1,500. Both options expire in one month, and the investor collects a premium of Rs. 100 per share.

If the price of HDFC Bank stock remains relatively stable and does not move significantly in either direction before the options expire, the seller of the Straddle can keep the entire premium of Rs. 100 per share. However, if the price of HDFC Bank stock moves significantly in either direction, the seller of the Straddle can face significant losses.

If the price of HDFC Bank stock goes up to Rs. 1,700, the buyer of the call option can exercise the option and buy the stock at Rs. 1,500, then immediately sell it at the market price of Rs. 1,700, resulting in a profit of Rs. 200 per share. The seller of the Straddle would be required to sell the stock at the strike price of Rs. 1,500, resulting in a loss of Rs. 200 per share, in addition to the premium collected.

Similarly, if the price of HDFC Bank stock goes down to Rs. 1,300, the buyer of the put option can exercise the option and sell the stock at Rs. 1,500, then immediately buy it at the market price of Rs. 1,300, resulting in a profit of Rs. 200 per share. The seller of the Straddle would be required to buy the stock at the strike price of Rs 1,500, resulting in a loss of Rs. 200 per share in addition to the premium collected.

In essence, the risk of selling a straddle is unlimited, as the price of the underlying asset can move significantly in either direction. The potential profit is limited to the premium collected from the buyer.

When selling straddles, it's important to consider the potential risks and rewards carefully. It's also important to have a solid understanding of the underlying asset and its potential for volatility. Conservative traders may prefer to sell straddles on stable, low-volatility assets, while more aggressive traders may choose to sell straddles on assets with higher volatility.

Example of Selling a Bank Nifty Option Straddle:

Let's say that a trader wants to sell a straddle on the Bank Nifty index, which is currently trading at 35,000. The trader decides to sell a call option and a put option with a strike price of 35,000 and an expiration date of one month. The trader collects a premium of Rs. 500 per share.

If the price of the Bank Nifty index remains relatively stable and does not move significantly in either direction before the options expire, the trader can keep the entire premium of Rs. 500 per share.

However, if the price of the Bank Nifty index moves significantly in either direction, the trader can face significant losses. If the price of the Bank Nifty index goes up to 36,000, the buyer of the call option can exercise the option and buy the index at 35,000, then immediately sell it at the market price of 36,000, resulting in a profit of Rs. 1,000 per share. The trader would be required to sell the index at the strike price of 35,000, resulting in a loss of Rs. 1,000 per share in addition to the premium collected.

Similarly, if the price of the Bank Nifty index goes down to 34,000, the buyer of the put option can exercise the option and sell the index at 35,000, then immediately buy it at the market price of 34,000, resulting in a profit of Rs. 1,000 per share. The trader would be required to buy the index at the strike price of 35,000, resulting in a loss of Rs. 1,000 per share in addition to the premium collected.

Technologies and Tools Used

Broker APIs

Broker APIs, or Application Programming Interfaces, are interfaces provided by online brokers that allow developers to integrate their trading platforms with external applications or software. These APIs give developers access to the broker's trading data, such as stock quotes, trade history, and order book information, as well as the ability to place trades and execute orders.

Broker APIs have become increasingly popular among traders and investors as they allow for a more streamlined and automated trading experience. With access to real-time market data and the ability to execute trades directly from their own software, traders can more efficiently manage their portfolios and respond to changing market conditions.

There are several different types of broker APIs available, each with its own unique features and capabilities. Some APIs provide basic functionality, such as access to real-time stock quotes and historical data, while others allow for more advanced trading strategies and automation.

One popular use of broker APIs is in the development of algorithmic trading systems. These systems use complex algorithms to analyze market data and execute trades based on predetermined rules and parameters. By leveraging broker APIs, developers can access real-time market data and execute trades automatically without the need for manual intervention.

Another common use of broker APIs is in the development of trading bots. These bots are automated trading systems that execute trades based on predetermined strategies and criteria. With access to real-time market data and the ability to execute trades directly from the bot's software, traders can more effectively manage their portfolios and respond to changing market conditions.

In addition to algorithmic trading and trading bots, broker APIs are also used in a variety of other applications and software. For example, some financial analysis software tools integrate with broker APIs to provide real-time market data and analysis. Similarly, some portfolio management tools use broker APIs to execute trades and manage portfolios directly from their software.

One of the key benefits of using broker APIs is the ability to access real-time market data and execute trades directly from external software. This can save traders significant amounts of time and improve their overall trading efficiency. Additionally, the use of broker APIs can provide more granular control over trades, allowing for more advanced trading strategies and risk management.

However, there are also some potential risks and drawbacks associated with the use of broker APIs. For example, poorly implemented or unreliable APIs can result in lost trades or other issues. Additionally, the use of APIs can introduce additional security risks, such as the potential for unauthorized access to trading accounts.

Overall, broker APIs are a powerful tool for traders and investors looking to streamline their trading process and improve their overall trading efficiency. However, it's important to carefully consider the potential risks and drawbacks associated with their use and to implement proper risk management and security protocols.

VS Code

Visual Studio Code is a free, open-source code editor developed by Microsoft, designed to be highly customizable and flexible, allowing developers to tailor it to their needs and workflows.

One of the key features of VS Code is its built-in support for a wide range of programming languages, including popular languages such as Python, Java, and JavaScript. The editor includes many features commonly found in more full-featured integrated development environments (IDEs), such as syntax highlighting, code completion, and debugging tools.

Another key feature of VS Code is its extensibility. The editor includes a robust extension marketplace, which allows developers to easily download and install extensions that add additional functionality to the editor. These extensions can range from simple utility tools, such as code formatting and linting, to more complex tools, such as Git integration and integrated terminals.

VS Code also includes a number of productivity features that can help developers work more efficiently. Its editor includes a built-in search function that can search across multiple files and directories and a powerful command palette that allows developers to quickly access and execute commands.

One of the main benefits of using VS Code is its flexibility and customizability. The editor can be tailored to meet the needs of developers working in a wide range of environments and on a wide range of projects. Additionally, because VS Code is open-source, developers can contribute to the development of the editor and create their own extensions and plugins.

However, there are some potential drawbacks to using VS Code. Because the editor is highly customizable, it can take some time to set up and configure to meet the specific needs of each developer. Additionally, the editor may not be as well-suited for larger projects or more complex development workflows as full-featured IDEs such as Eclipse or Visual Studio.

It is a powerful and flexible code editor that can be a valuable tool for developers working on a wide range of projects. Its customizability and extensibility make it a popular choice among developers, and its support for a wide range of programming languages and productivity features can help developers work more efficiently and effectively.

Python 3.10

Python 3.10 is the latest stable release of the Python programming language, which was released in October 2021. It includes several new features and improvements over the previous versions of Python, making it a popular choice among developers.

One of the key features of Python 3.10 is improved performance. The new version includes a number of optimizations that help to make Python programs run faster, including faster parsing and improved garbage collection. Additionally, the new version includes several new built-in functions, including a new syntax for combining two dictionaries and a new method for removing prefixes and suffixes from strings.

Another significant feature of Python 3.10 is improved support for type annotations. Type annotations allow developers to specify the type of a variable, parameter, or return value in their code. This helps to make the code more self-documenting and can improve code quality by catching type-related bugs at compile time. Python 3.10 includes several improvements to the type annotation system, including better support for generics and improved error messages.

Python 3.10 also includes several new features that make it easier to work with async programming. Async programming allows developers to write code that can perform multiple tasks simultaneously, making it ideal for web development, networking, and other applications where concurrency is important. The new version includes several new modules and functions that make it easier to work with async programming, including improved support for asyncio and new syntax for defining async functions.

Finally, Python 3.10 includes several other improvements and optimizations, including improved error messages, better support for Unicode, and improved compatibility with other languages and platforms. Overall, Python 3.10 is a significant update to the language that brings many new features and improvements that make it an even better choice for a wide range of applications.

Python Libraries

1. The `requests` library is a popular Python library for making HTTP requests. It provides a simple and intuitive way to interact with web services, allowing you to easily make HTTP requests and handle responses. With `requests`, you can send GET, POST, PUT, and DELETE requests, add custom headers and cookies, and handle errors and redirects. It's a great library for working with RESTful APIs and web scraping.

2. The `datetime` library is a built-in Python library for working with dates and times. It provides a number of classes and functions for representing and manipulating dates and times, including `datetime`, `date`, and `time`. With `datetime`, you can perform common operations like adding and subtracting dates, comparing dates, and formatting dates and

times for display. It's a useful library for any application that needs to work with dates and times.

3. The `logging` library is a built-in Python library for logging messages from your application. It provides a simple way to write messages to a log file or console, with support for different log levels and formatting options. With `logging`, one can easily debug issues in your application by writing detailed log messages that can be analyzed later.

4. The `time` library is a built-in Python library for working with time values. It provides a number of functions for working with time, including `time()`, `sleep()`, and `strftime()`. With `time`, you can perform operations like measuring the time it takes to execute code, sleeping for a certain amount of time, and formatting time values for display.

5. The `yaml` library is a Python library for working with YAML data. YAML is a human-readable data serialization format that's often used for configuration files and data exchange between applications. With `yaml`, you can easily read and write YAML data in Python, making it a great library for working with configuration files and other YAML-based data sources.

6. The `pandas` library is a popular Python library for data analysis and manipulation. It provides data structures like `Series` and `DataFrame` that make it easy to work with tabular data in Python. With `pandas`, you can perform operations like filtering, grouping, and aggregating data, as well as visualizing data with built-in plotting functions. It's a great library for working with large datasets and performing data analysis tasks.

7. The `pyotp` library is a Python library for generating and verifying one-time passwords (OTP). OTPs are often used for two-factor authentication (2FA), where users must enter a code in addition to their password to log in to a system. With `pyotp`, you can easily generate and verify OTPs in your Python application, making it a great library for building secure login systems.

8. The `os` library is a built-in Python library for interacting with the operating system. It provides a number of functions for working with files, directories, and other operating system resources. With `os`, you can perform operations like creating and deleting files and directories, listing the contents of directories, and working with environment variables.

9. The `urllib` library is a Python library for working with URLs and HTTP requests. It provides a number of modules for working with different parts of the URL, including `urllib.request` for making HTTP requests, `urllib.parse` for parsing URLs, and `urllib.error` for handling errors. With `urllib`, you can easily make HTTP requests and work with URLs in your Python application.

10. The zipfile module in Python provides a way to read and write ZIP files. It can be used to compress and decompress files, as well as extract files from a ZIP archive. The zipfile module is particularly useful when working with large archives, as it provides a way to read and write files in chunks rather than loading the entire archive into memory at once.

11. The timeit module in Python provides a way to measure the execution time of small code snippets. It can be used to compare the performance of different algorithms, test the speed of specific functions, and optimize code. The timeit module is particularly useful when optimizing code, as it provides a way to measure the impact of small changes on the overall performance of a program.

Telegram

A Telegram bot can be a useful tool for getting live updates of the execution and alerts in a trading strategy. The bot can be integrated with the Python script to send messages to a Telegram channel or group, providing real-time updates on the progress of the strategy. The bot can be configured to send alerts when specific events occur, such as when a trade is executed or when certain market conditions are met. This can be particularly useful for traders who need to stay on top of market movements and want to be notified immediately when a trading opportunity arises. With a Telegram bot, traders can have instant access to important information and take action quickly, giving them an edge in the market.

Data Connection and Broker Details

As a part of my algorithmic trading project, I have chosen to use Finvasia as my brokerage firm for its zero brokerage fees and active API support. The availability of proper documentation and examples, as well as live prices through WebSocket, made it a suitable choice for my trading needs.

Finvasia is a fast-growing fintech company in India that offers various financial services, including zero brokerage, zero clearing, zero account opening, zero AMC, and more. The company also serves as a one-stop-shop for Foreign Portfolio Investors (FPI) looking to invest in the Indian markets and has advised institutional clients on their investments in fourteen countries.

The team at Finvasia is led by ex-Wall Street professionals with deep financial expertise who aim to make financial tools accessible and affordable for investors. The company received FDI funding from notable venture capitalists in 2016, valuing it at INR 1.5 billion. This funding enabled Finvasia to achieve its mission of cutting trading costs and offering technology-driven financial services to its clients.

With zero brokerage fees, I was able to save significantly on transaction costs, which is a significant advantage for an algorithmic trader. Moreover, Finvasia's active API support was crucial to the success of my trading strategy. The API was well-documented with examples, making it easy for me to integrate it with my trading platform. The live prices provided through WebSocket also helped me make quick and informed trading decisions.

In addition to these advantages, Finvasia offers a range of other services that make it a compelling choice for traders. The company provides an online trading platform, enabling traders to access real-time market data and execute trades. They also offer a mobile trading app, making it easy to trade on the go. The platform supports various asset classes, including equities, commodities, currencies, and more.

The account opening process with Finvasia was straightforward and easy to navigate. The company offers a paperless account opening process, making it convenient for traders to open an account from anywhere. They also provide a free demat account and a 2-in-1 account that allows traders to trade in both equity and commodity segments.

In conclusion, choosing Finvasia as my brokerage firm has been a great decision for my algorithmic trading project. The zero brokerage fees and active API support, coupled with the range of other services and asset classes offered, make it an excellent choice for traders looking to execute their trading strategies efficiently and effectively. The team's financial expertise and vision of making financial tools accessible and affordable for investors also give me confidence in their ability to provide quality financial services.

Telegram Connection

For my trading project, I have utilized a Telegram connection for receiving live updates. The code for the Telegram bot is implemented in Python, and the necessary credentials are stored in a YAML file.

```
with open('telegram_bot.yml') as f:
   bot = yaml.load(f, Loader=yaml.FullLoader)
bot_token, chat_id  = bot['bot_token'] , bot['chat_id']
def send_message(text):
   base_url =
f'https://api.telegram.org/bot{bot_token}/sendMessage?chat_id={chat_id}&text={text}'
   requests.get(base_url)
msg = f"Bot Connected at %s" % (datetime.now().strftime("%H:%M:%S"))
send_message(msg)
```

To start with, I have imported the YAML module and read the credentials from the YAML file using the `load` method. The bot token and chat ID are used to initialize the bot object.

I have defined a function `send_message(text)` to send messages to the Telegram chat. The function takes the text message as input and constructs a URL using the bot token, chat ID, and message text. The `requests. get()` method is used to send a GET request to the Telegram API, which in turn sends the message to the specified chat.

Finally, I have used the `send_message()` function to send a message to the Telegram chat. In this case, I have sent a message stating that the bot has been successfully connected, along with the current timestamp.

The purpose of this code is to log the status of the bot and receive live updates on the trading activity. By sending messages to the Telegram chat, I can keep track of the bot's performance, as well as any errors or issues that may arise during the trading process.

This feature is particularly useful for automated trading systems, where it is essential to monitor the bot's activity and performance continuously. With the Telegram connection in place, I can receive real-time updates and take prompt action if necessary.

In conclusion, the Telegram connection for live updates is a valuable feature for any trading project. It enables real-time monitoring of the bot's activity and performance, which is crucial for automated trading systems. By using the code provided above, I can easily integrate the Telegram connection into my trading project and stay informed of any developments in real-time.

Modules Used

The `Logger` module is a built-in module in Python that allows developers to write custom log messages in their code for debugging purposes. It provides a way to collect and store log messages in a structured manner, which can be used to analyze issues and improve the codebase. The code sets up a basic logging configuration with a specified log level, date format, message format, and log file handler. If the 'logs' directory does not exist, it creates one. The `FileHandler` specified in the code writes log messages to a new file with a name containing the current date in 'YYYY-MM-DD' format, prefixed with 'straddle-'.

The Telegram bot module involves setting up a Telegram bot by loading the bot token and chat ID from a YAML file. It also has a `send_message()` function to send updates to the Telegram channel. The function takes a message text and sends it to the channel using the Telegram bot API.

This code block commented the start of the program that uses the ShoonyaApiPy API. It first creates an instance of the ShoonyaApiPy class. It then reads in the login credentials from a YAML file using the PyYAML module. The credentials include the user ID, password, two-factor authentication token, vendor code, API secret key, and IMEI number. The `api.login()` method is then called with these credentials as arguments to authenticate the user and establish a session with the API. The return value is stored in the `ret` variable.

The WebSocket code include establishing and handling a websocket connection using ShoonyaApiPy library. The code defines two event handlers - `event_handler_feed_update` and `event_handler_order_update` - which are called whenever there is an update in the tick data or order data respectively. The `open_callback` function is called when the websocket connection is successfully established. The `startWebSocket` function starts the websocket connection and waits for it to open by continuously checking the value of `feed_opened` variable. Once the connection is established, the function returns `True`. Overall, the code is used to receive real-time updates for stock prices and order status.

Then code then downloads market symbols for various exchanges in India and saves them to a folder called "Masters". It starts by creating the folder if it does not exist, then proceeds to download the zip files for each exchange and asset type. It then extracts the contents of each zip file to the "Masters" folder, and deletes the original zip file. After all the files have been downloaded and extracted, it logs the successful download of all the masters and sends a message via Telegram to indicate that the masters have been updated.

The `get_expiry_dates` function retrieves all the expiry dates for the given symbol by querying a pandas dataframe named `NFO_SYMBOLS`. It extracts the dates by filtering out the rows with the matching symbol, parsing the date string to the `datetime` object, sorting the list, and returning the dates in the required format.

The `get_current_expiry` function uses `get_expiry_dates` function to retrieve all the expiry dates for the symbol and returns the current expiry date.

The `get_next_expiry` function also uses `get_expiry_dates` function to retrieve all the expiry dates for the symbol and returns the next expiry date.

The `get_strike_diff` function retrieves all the strikes for the given symbol from the `NFO_SYMBOLS` dataframe and calculates the difference between consecutive strikes. It returns the minimum of all the calculated differences.

The `get_lot_size` function takes trading symbol and returns its lot size. It retrieves the lot size from the `NFO_SYMBOLS` dataframe by filtering out row with matching trading symbol.

The `get_symbol_details` function uses the `get_expiry_dates` and `get_strike_diff` functions to retrieve current and next expiry dates and strike difference for the given symbol. It logs the message and sends it to the specified chat application.

The `get_atm_strike` function takes a token and calculates the strike price closest to the last traded price for the given token. It returns the calculated strike.

The `get_NFO_token` function takes a trading symbol and retrieves its corresponding NFO token from the `NFO_SYMBOLS` dataframe.

The `expiry` function calculates the current expiry date of the given token by querying the API for the current ATM call and put options. If the sum of their last traded price is greater than 2.33 times the strike difference and both options have last traded prices greater than 1.11 times the strike difference, then it returns the current expiry date, else it returns the next expiry date.

The `get_atm` function uses `expiry` and `get_atm_strike` functions to calculate the ATM strike and call and put option symbols for the given token and symbol. It logs the message and returns the calculated values.

The `straddle_shift_required` function takes a strike price and a token and checks if the last traded price for the given token is 1.44 times the strike difference away from the given strike price. If it is, then it returns True, else it returns False.

The `get_fillprice` function takes a norenordno and retrieves its corresponding fill price from the API.

The `get_openorder` function takes a norenordno and retrieves its corresponding status from the API.

The `get_daily_bnf_mtm` function retrieves the daily mark to market (MTM) for the BankNifty index by querying the API for the current positions.

The `straddle` function takes two optional parameters - `token` and `symbol`, and returns the strike price of the at-the-money (ATM) option, along with the symbols for the call option and put option. The function first calls the `get_atm` function to retrieve the ATM strike price and option symbols. It then calculates the lot size and quantity for each option based on the `LOTS` constant and the lot size retrieved from the `get_lot_size` function. It places a sell market order for both the call and put options using the `place_order` function of the `api` object, and subscribes to their respective token streams using the `subscribe` function. It then retrieves the order numbers and fill prices using the `get_fillprice` function, and logs and sends messages with the relevant information.

The `current_straddle_exit` function takes two symbol parameters - `CE_SYMBOL` and `PE_SYMBOL` - representing the call and put options of the current straddle position. It places buy market orders for both options using the `place_order` function, unsubscribes from their token streams using the `unsubscribe` function, retrieves the order numbers and fill prices using the `get_fillprice` function, and logs and sends messages with the relevant information. The function is used to exit the current straddle position.

Strategy Explanation

The options market is a complex market with many different strategies that can be used. One of the most popular options trading strategies is the Straddle Strategy. The Straddle Strategy involves buying or selling an at-the-money call option and an at-the-money put option at the same time. The Straddle buying Strategy is a good strategy to use when you believe that there will be a significant price movement in the underlying asset, but you are not sure which direction the price will move. However, if market seems stable and price is not expected to move in either direction, then Straddle can be sold and the premium can be pocketed.

The Straddle selling strategy is based on the following algorithm:

1. Define all the functions and constants.

2. Select the instruments to be traded.

3. Check if a Straddle is open. If not, open a Straddle using the Straddle function, by selling at-the-money options. The Straddle function returns the strike and symbol details and counts it as one trade with a target of Rs. 1000 and a trailing stop loss of Rs. 1000 from the best mark-to-market (MTM).

4. When the target of Rs. 1000 hits, check if the new Straddle price that needs to be executed is the same Straddle. If so, instead of booking the target, increase the target by Rs. 250.

5. Keep checking for exit conditions, i.e., if the target has been hit or the stop-loss has been hit. If neither condition is met, check if the Straddle needs to be shifted using the `Straddle_shift_required` function. If the condition is true, shift the Straddle to a new strike price. No new trade is counted in adjustments, and a new trade starts only when either the stop-loss or target gets hit.

6. The final exit happens when the time is 3:29:55 pm, exactly four seconds before the market closes, so the Straddle exit gets executed correctly. The daily maximum trades are five, and the maximum stop losses to be taken in a day are two, i.e., a maximum potential loss of Rs. 2000 and a maximum potential profit of the premiums collected.

7. The execution also logs everything with the logger and sends all the important updates to Telegram, including the trade execution, entry exits, adjustments, etc.

The algorithm uses a while loop that keeps running until the daily maximum trades or maximum stop-losses have been reached. The loop checks the mark-to-market (MTM) value of the Straddle at each iteration and performs the necessary actions depending on the MTM value.

The code uses the Finvasia's API to fetch the MTM value, which is then compared with the stop-loss and target values. If the stop-loss value is hit, the code sends a message to Telegram and closes the Straddle. If the target value is hit, the code checks if the new Straddle price is the same as the old Straddle. If it is, the code increases the target value by Rs. 250. If the new Straddle price is different, then the trade Target gets hit and another straddle is opened and counted as new trade. The algorithm also includes a function to check if a Straddle shift is required. The Straddle_shift_required function checks the MTM value of the Straddle and determines if the price movement requires the Straddle to be shifted to a new strike price. If the condition is true, the Straddle is shifted to the new strike price using the Straddle_shift function. The Straddle_shift function calculates the new strike price based on the current market price and shifts the Straddle to the new strike price by selling at-the-money options.

Overall, this Algo Trading strategy using the Straddle Strategy is designed to automate the process of trading options in the Indian markets. The strategy uses Python programming language and relies on several third-party libraries, such as Finvasia's API, to interact with the Indian markets. The algorithm is designed to execute trades automatically using computer programs and algorithms, and includes functions to open and close Straddles, shift Straddles to new strike prices, and check exit conditions. The strategy also includes logging and messaging features to keep the trader updated on the status of their trades.

Performance in Live Market

During the month of April, The algorithm was used to execute trades in the financial markets. The algorithm had a capital requirement of 175,000 rupees and was active for a total of 15 trading days.

Out of these 15 trading days, the algorithm was able to generate profits on 9 of them, resulting in a win rate of 60%. The remaining 6 days resulted in losses. The maximum winning streak was 3 days, while the maximum losing streak was 2 days.

In terms of overall performance, the algorithm was able to generate a total profit of 5,926.25 rupees during the month of April, resulting in a return on investment (ROI) of 3.39%. The maximum profit earned on a single trading day was 2,357.5 rupees, while the maximum loss on a single trading day was -1,041.25 rupees.

On average, the algorithm generated a profit or loss of 395.08 rupees per trading day. On days where a profit was earned, the average profit was 916.39 rupees, while on days where a loss was incurred, the average loss was -386.88 rupees.

The maximum drawdown, which represents the largest peak-to-trough decline in the algorithm's equity, was 0 rupees during the month of April. This means that the algorithm did not experience any significant losses during the month.

Looking at the data, it is clear that the algorithm was able to generate consistent profits during the month of April. The win rate of 60% indicates that the algorithm was able to accurately identify profitable trading opportunities more often than not. Additionally, the fact that the algorithm did not experience any significant drawdowns during the month suggests that it was able to manage risk effectively.

In addition to the performance of the algorithm, it is also important to consider the broader context of the financial markets during the month of April. During this time, the markets experienced significant volatility due to a variety of factors, including geopolitical tensions and changes in monetary policy. Despite these challenges, the algorithm was able to generate consistent profits, which is a testament to its effectiveness in navigating difficult market conditions.

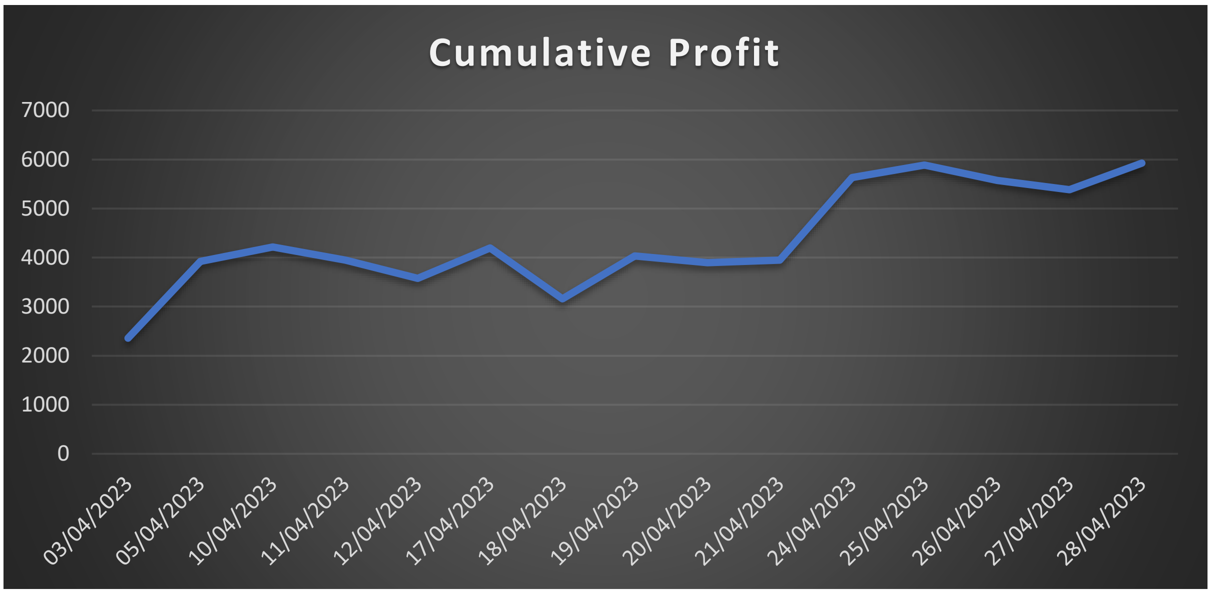
Overall, the performance of the algorithm during the month of April was impressive, and it serves as a strong foundation for further optimization and refinement in the future. With the right approach and continued effort, it is possible to generate consistent profits.

Profit/Loss on Each Trading Day :-

Dates	Profit	Charges	Net Amount
03/04/2023	2357.5	48.91	2308.59
05/04/2023	1566.25	45.01	1521.24
10/04/2023	293.75	185.66	108.09
11/04/2023	-267.5	102.45	-369.95
12/04/2023	-370	39.42	-409.42
17/04/2023	617.5	198.27	419.23
18/04/2023	-1041.25	101	-1142.25
19/04/2023	878.75	29.88	848.87
20/04/2023	-141.25	46.53	-187.78
21/04/2023	55	99.79	-44.79
24/04/2023	1685	77.46	1607.54
25/04/2023	253.75	75.49	178.26
26/04/2023	-312.5	55.15	-367.65
27/04/2023	-188.75	134.22	-322.97
28/04/2023	540	117.34	422.66
Total	**5926.25**	**1356.58**	**4569.67**

Overall Performance Statistics :-

S. No.	Statistics	Value
1	Capital Required (Max)	175000
2	Total Trading Days	15
3	Win Days	9
4	Loss Days	6
5	Max Winning Streak Days	3
6	Max Losing Streak Days	2
7	Win Rate	60.00%
8	Total Profit	5,926.25
9	ROI	3.39%
10	Max Profit in a Day	2357.5
11	Max Loss in a day	-1041.25
12	Avg Profit/Loss Daily	395.08
13	Avg Profit on Profit Days	916.39
14	Avg Loss on Loss Days	-386.88
15	Max Drawdown	0
16	Max Drawdown %	0.00%


Cumulative Profit
7000
6000
5000
4000
3000
2000
1000
0
03/04/2023
05/04/2023
10/04/2023
11/04/2023
12/04/2023
17/04/2023
18/04/2023
19/04/2023
20/04/2023
21/04/2023
24/04/2023
25/04/2023
26/04/2023
27/04/2023
28/04/2023

Future Prospects

The BankNifty Straddle Selling strategy has shown promising results, and as with any strategy, there is always room for improvement. Here are some potential future prospects that could be implemented to enhance the strategy:

1. Adjusting the strategy to perform better in low volatility conditions: One way to improve the strategy is by modifying it to perform better in low volatility conditions. This could be done by quickly adjusting straddles and holding straddles in high volatility times. This would require monitoring market conditions closely and adapting the strategy accordingly.

2. Making the strategy more dynamic: Another potential improvement to the strategy is making it more dynamic. This could be achieved by automatically stopping trades when market conditions are no longer favorable. By doing so, losses can be minimized and profits can be maximized.

3. Integrating a GUI dashboard and controller: Implementing a GUI dashboard and controller would provide a better view of the strategy's performance and enable easier monitoring of trades. This would not hinder the performance of the algorithm but would provide better insights to make informed decisions.

4. Running the straddle entry and exit order asynchronously: Running the straddle entry and exit order asynchronously can improve execution speed and reduce slippage. This can be achieved by using multithreading in the code, allowing for the strategy to execute more efficiently.

5. Using object-oriented concepts: Using object-oriented concepts can make the code cleaner, more organized, and easier to maintain. By structuring the code this way, it can be easier to implement changes and fix errors as the strategy evolves.

6. Auto-selecting position sizing: Implementing an auto-selecting position sizing feature based on capital available and risk-taking capacity can help reduce losses and maximize profits. This would ensure that the position size is always appropriate for the given market conditions.

7. Adding an emergency stop button: Adding an emergency stop button to exit all open positions and stop the code execution can provide a safety net in case of unexpected market conditions or errors in the code. This can help minimize losses and prevent further damage.

8. Improving error handling: Extensively improving error handling in the code can help reduce downtime and ensure the strategy runs smoothly. By implementing thorough error handling procedures, the strategy can be more robust and less prone to failures.

9. Integration of machine learning algorithms: Incorporating machine learning algorithms into the strategy can help to better predict market conditions and make more informed trading decisions. By analyzing historical market data, the algorithm can learn to recognize patterns and signals that indicate when to enter or exit trades.

10. Expansion to other markets: While the BankNifty market may be the focus of the current strategy, there is potential to expand the algorithm to other markets. By adapting the algorithm to work with other indexes or stocks, the strategy can be diversified to mitigate risk and potentially increase profitability.

In conclusion, the BankNifty Straddle Selling strategy has demonstrated promising results and has room for further improvement and expansion. As market conditions continue to evolve, incorporating new technology and refining the algorithm will be crucial to maintaining its effectiveness. By remaining adaptable and innovative, this strategy has the potential to continue generating profitable returns in the future.

References

Code for this strategy – https://github.com/AAAI-Lab/Sachin-Sharma

1. Python Software Foundation. (n.d.). The Python Language Reference. Retrieved from https://www.python.org/doc/

2. Shoonya. (n.d.). API Documentation. Retrieved from https://www.shoonya.com/api

3. Shoonya. (n.d.). Shoonya Python API. Retrieved from https://www.shoonya.com/static/website/pdf/shoonya-Python-API.pdf

4. Shoonya. (n.d.). API Documentation. Retrieved from https://www.shoonya.com/api-documentation

5. Shoonya. (n.d.). NFO_symbols.txt. Retrieved from https://api.shoonya.com/NFO_symbols.txt.zip

6. Shoonya-Dev. (n.d.). ShoonyaApi-py. GitHub. Retrieved from https://github.com/Shoonya-Dev/ShoonyaApi-py

7. Microsoft. (n.d.). Visual Studio Code Documentation. Retrieved from https://code.visualstudio.com/docs

8. Nagarvani, G. (2021, March 25). Deploying Algorithmic Trading Strategy: Part 3-Telegram Updates. Medium. Retrieved from https://medium.com/@ganeshnagarvani/deploying-algorithmic-trading-strategy-part-3-telegram-updates-b22cd101a258

9. OpenAI. (n.d.). OpenAI Chat. Retrieved from https://chat.openai.com/

10. Zerodha. (n.d.). Varsity. Retrieved from https://zerodha.com/varsity/