Department of Aerospace Engineering

INDIAN INSTITUTE OF TECHNOLOGY, BOMBAY

Dynamic Stock Market Analysis

Winter in Data Science(WIDS)

Anish Singh - 23M0016

M.tech: Dynamics And Control– Aerospace Department

Contents

List of Figures

1 Introduction

The primary objective of this project is to design and develop a sophisticated NIFTY OPTIONS trading bot using the Python programming language. The bot will be equipped to leverage real-time NIFTY data and employ analysis to generate actionable trading signals. The bot will integrate various technical indicators to achieve this, drawing on established metrics such as moving averages and candlestick patterns. These indicators and patterns will contribute to the bot's ability to assess market trends, momentum, and potential entry or exit points. One key aspect of innovation in this project involves incorporating additional features to enhance the accuracy of the bot analysis. These features use machine learning algorithms to adapt and optimize trading strategies based on changing market conditions.

In order to validate and fine-tune the effectiveness of the trading bot, a crucial step involves backtesting on historical market data. This process allows us to simulate the bot's performance under realistic market conditions, evaluating its ability to generate profitable signals and adapt to historical trends. The development process will also involve considerations for risk management and robustness. Implementing safeguards and risk mitigation strategies will be integral to ensure the bot operates within predefined risk parameters, preventing large losses during adverse market conditions. At last backtesting is done on the real market trade.

In summary, this comprehensive project aims to create a Python-based stock market trading bot that utilizes real-time stock data and proprietary analysis, integrates innovative features, and undergoes rigorous backtesting for robust performance in dynamic market conditions.

2 Algorithm

2.1 Libraries and Modules

The Algorithm performs several tasks, including data preprocessing, feature engineering, machine learning model training, real-time data fetching, and trading decision-making based on the predictions and options data.It starts by importing the necessary libraries, including requests, tvDatafeed, pandas4, talib, SimpleImputer, and datetime. It then reads a CSV file containing NIFTY data using the $pd.read\_csv()$ function and preprocesses the data by converting the datetime column to a datetime object using the $pd.to\_datetime()$ function. The script then engineers new features such as moving averages, MACD, and various candlestick patterns using the talib library. All the libraries used are mentioned below:

requests: Used to send HTTP requests to a URL.
tvDatafeed: It seems to be a library for fetching financial data from TradingView.
pandas: A powerful data manipulation library.
talib: Technical Analysis Library, used for calculating technical indicators.
SimpleImputer: Used for imputing missing values in the dataset.
datetime: For handling date and time.
MinMaxScaler: Used for normalizing data.
RandomForestRegressor: Part of scikit-learn, used for training a random forest regression model.
shuffle: Used to shuffle data.

2.2 Reading and Preprocessing Historical Data

Reading and preprocessing historical data is a crucial step in developing trading algorithms or conducting financial analysis. Here, historical data for the NIFTY index is read from a CSV file and then subjected to various preprocessing steps. Here's an explanation of the key concepts involved in reading and preprocessing historical financial data:

1. **Reading Historical Data:** The data is typically stored in a CSV (Comma-Separated Values) file, a common format for tabular data.The data is loaded into a Pandas DataFrame, a powerful data manipulation library in Python. DataFrames provide a convenient structure for handling and analyzing tabular data.

2. **Data Exploration** Analysts often start by examining the structure and content of the dataset. This includes checking the number of rows and columns, data types, and the first few rows of data. Relevant columns are selected based on the analysis requirements. In the provided code, columns like 'datetime', 'open', 'high', 'low', 'close', etc., are likely important for financial analysis.

3. **Data Cleaning:** Missing data is common in real-world datasets. The ‘SimpleImputer‘ from the scikit-learn library fills in missing values. In this case, the strategy chosen ('mean') determines how missing values are imputed. Columns that are not needed for analysis can be dropped to simplify the dataset.

4. **Feature Engineering:** The 'datetime' column is converted to a datetime format, allowing for easier manipulation and analysis of time-series data. Additional features like 'year', 'hour', and 'minute' are derived from the datetime information. These features can be useful for time-based analysis.

5. **Technical Indicators:** The historical price data calculates technical indicators like the Exponential Moving Average (EMA). These indicators help capture trends and patterns in the market. MACD (Moving Average Convergence Divergence) is another widely used indicator calculated by taking the difference between two moving averages of an asset's price. It is used to identify potential trend changes and provide buy or sell signals.

6. **Candlestick Patterns:** Candlestick patterns, such as engulfing, hammer, and shooting star, are identified using the price data. These patterns are often used in technical analysis to predict future price movements.

7. **Data Transformation:** The data is often normalized or scaled to bring all features to a similar scale. This is important for machine learning algorithms, ensuring that no particular feature dominates the others.

8. **Data Subset Selection:** The code selects a subset of the data, starting from the 201st row. This subset is then reset to have a new index starting from 0. This step might be done to exclude the initial data points or to align with a specific starting point for analysis.

   These preprocessing steps aim to prepare the historical data for further analysis, including machine learning model training, technical analysis, or any other financial modeling techniques. Each step is designed to clean, enhance, and structure the data to derive meaningful insights and build robust models.

2.3 Training Machine Learning Model

The key concepts and steps involved in this process:

1. **Data Preprocessing:** The historical stock market data is preprocessed to extract relevant features. This includes calculating technical indicators such as Exponential Moving Average (EMA), Moving Average Convergence Divergence (MACD), and candlestick patterns. Missing values in the dataset are handled using an imputer. In this case, the ‘SimpleImputer‘ is used to replace missing values with the mean.

2. **Normalization:** The data is normalized using a 'MinMaxScaler'. Normalization is a crucial step in machine learning where features are scaled to a standard range (usually between 0 and 1). This ensures that all features contribute equally to the training process.

3. **Random Forest Regression Model:** The machine learning model chosen for this task is a Random Forest Regressor. A random forest is an ensemble learning method that constructs a multitude of decision trees during training and outputs the average prediction of the individual trees for regression tasks.

4. **Training:** The model is trained on the preprocessed and normalized data. The input features are the various technical indicators and other relevant data points, while the target variable is the predicted outcome. The number of trees in the forest ('n_estimators') is set to 60, and a random state is specified for reproducibility.

5. **Shuffling Data:** The training data is shuffled using the 'shuffle' function from scikit-learn. Shuffling the data helps prevent any bias that might arise if the data is ordered in a specific way.

6. **Model Evaluation:** It's crucial to evaluate the performance of the trained model. However, the code snippet doesn't include explicit model evaluation steps, such as splitting the data into training and testing sets and assessing metrics like Mean Squared Error (MSE) or R-squared.

7. **Predictions:** Once the model is trained, it can be used to predict new data. Predictions ('y_pred') are generated in the code for a specific set of input features ('x_pred'), representing the current market conditions.

8. **Model Interpretation:** Understanding the importance of different features in the model can be essential for model interpretation. In the case of a Random Forest, feature importance can be examined to gauge which features contribute more to the predictions.

9. **Deployment Considerations:** The trained model can be further integrated into a larger trading system or algorithm. Considerations for model deployment, monitoring, and periodic retraining are crucial for ensuring the model's ongoing effectiveness in evolving market conditions.

   It's important to note that successful machine learning in financial markets requires a deep understanding of the chosen algorithms' domain and limitations. Additionally, thorough backtesting and validation are necessary to accurately assess the model's performance before deploying it in real trading scenarios.

2.4 Fetching real-time data

Fetching real-time data is a crucial aspect of algorithmic trading, providing the most up-to-date information for making trading decisions. The provided code fetches the real-time data using the 'TvDatafeed' library,

which seems to be a custom library for accessing financial data from TradingView.
Here's a breakdown of the process of fetching real-time data:

Library Selection:

The code utilizes the 'TvDatafeed' library, which suggests that it is specifically designed to interface with TradingView's data feed. TradingView is a popular platform for technical analysis and charting.

Object Initialization:

An instance of the 'TvDatafeed' class is created using the line 'tv = TvDatafeed().'

Data Retrieval:

The 'get_hist' method is called on the 'TvDatafeed' instance to fetch historical data. Parameters such as 'symbol' ('NIFTY'), 'exchange' ('NSE' for the National Stock Exchange), 'interval' (time interval for each data point, e.g., 5 minutes), and 'n_bars' (the number of historical bars to retrieve) are provided.

Data Manipulation:

The fetched data is converted into a Pandas DataFrame ('df_n'), making it easier to work with and analyze. Additional data manipulation steps may follow, including extracting relevant features and technical indicators.

Time Series Analysis:

The real-time data, now in the form of a time series, can be analyzed to derive insights into market trends, price movements, and potential trading opportunities.

Integration with Machine Learning Models:

As the code shows, the fetched real-time data is integrated with machine learning models to predict future price movements.

Decision Making:

Trading decisions are made based on real-time data analysis and predictions. This specific code implements options trading strategies based on the predicted movements.

Continuous Updates: For an algorithmic trading system, fetching real-time data is typically done in a loop or on a regular schedule to ensure the model works with the latest available information.
It's important to note that fetching real-time data in a production trading system involves considerations such as data latency, reliability, and the source's terms of service. Additionally, developers often implement error handling and redundancy mechanisms to ensure the robustness of the data-fetching process.

2.5 Predicting and Trading:

In the provided code, the section labeled "Predicting and Trading" involves using a trained machine learning model to predict the future movement of a financial instrument, in this case, the NIFTY index. The

predictions are then used to inform trading decisions, specifically regarding options trading. Here's an overview of this process:

Machine Learning Model: A Random Forest Regression model is trained using historical stock market data. This model is designed to predict the future movement of the NIFTY index based on various features, including technical indicators like Exponential Moving Average (EMA), Moving Average Convergence Divergence (MACD), and candlestick patterns.The training involves using a dataset with labeled outcomes (in this case, gains or losses) to teach the model to recognize patterns that lead to profitable or unprofitable situations.

Prediction: The trained model is then used to predict the expected movement of the NIFTY index. The predictions are likely expressed as numerical values representing potential gains or losses.

Trading Strategies: The predictions are utilized to make decisions about trading options. Options are financial instruments that give the buyer the right (but not the obligation) to buy or sell an underlying asset at a predetermined price before or at the expiration date. Depending on the direction of the predicted movement (upward or downward), the code suggests implementing options trading strategies. For example, the code suggests buying call options if the prediction is positive (indicating an expected upward movement). Conversely, if the prediction is negative, it suggests buying put options.

Risk Management: The code implements some basic risk management by setting stop-loss and profit-taking levels. These levels are calculated based on the predicted movement and are crucial for controlling potential losses and securing profits.

Execution: The code may execute trades on the options market through an appropriate platform or API once the trading decisions are made based on the predictions.

Output and Explanation: The code generates output messages providing information about the predictions and the corresponding trading strategies. Explanations are given regarding the predicted movement, entry time, suggested stop-loss levels, and profit-taking levels.

It's important to note that trading in financial markets involves inherent risks, and past performance does not necessarily indicate future results. The success of any trading strategy depends on various factors, including market conditions, the accuracy of predictions, and effective risk management. Additionally, regulatory compliance and ethical considerations are critical when developing and deploying algorithmic trading systems. It's recommended to thoroughly test any strategy in a simulated environment before applying it to real trading situations.

2.6 Options Chain Analysis:

Options Chain Analysis involves examining the available options contracts for a particular financial instrument, such as a stock or index. In the context of the provided code, it appears to focus on the NIFTY index.

Here's an explanation of the key aspects of Options Chain Analysis:

Options Chain Overview: An options chain lists all available options contracts for a particular security. It typically includes both call options (giving the right to buy) and put options (giving the right to sell) with various strike prices and expiration dates.

Components of an Options Chain: The prices at which the option holder can buy (for call options) or sell (for put options) the underlying asset. The dates on which the options contracts expire. Options are typically categorized into different expiration cycles (e.g., monthly or weekly). Specifies whether the option is a call option (giving the right to buy) or a put option (giving the right to sell). The price at which an options contract is bought or sold in the market.

Implied Volatility and Historical Volatility: Represents the market's expectation of future price volatility. Higher IV often leads to higher option premiums. A measure of the past price fluctuations of the underlying asset. Traders may compare IV with HV to identify potential opportunities.

Open Interest and Volume: The total number of outstanding options contracts for a particular strike price and expiration date. High open interest suggests active trading.The number of contracts traded during a specific period. High volume indicates increased market interest.

Options Trading Strategies: Traders analyze options chains to develop various trading strategies, such as covered calls, protective puts, straddles, and strangles. Understanding the risk-reward profiles of different options strategies is crucial for effective decision-making.

Options Pricing Models: Black-Scholes and Binomial models are commonly used to estimate the fair value of options based on factors like the current stock price, strike price, time to expiration, volatility, and interest rates.

Risk Management: Traders assess the risk associated with options positions, considering factors like delta (sensitivity to stock price changes), gamma (rate of change of delta), theta (time decay), and vega (sensitivity to implied volatility changes).

Market Sentiment Analysis: Options chain data can provide insights into market sentiment. For example, many call options at a specific strike price may indicate bullish sentiment.

In summary, Options Chain Analysis is a critical aspect of derivatives trading. It involves studying the available options contracts to make informed decisions about potential price movements, volatility, and risk. Traders use this analysis to implement strategies that align with their market outlook and risk tolerance.

2.7 Explanation and Output

The code print information about the predicted movement of the NIFTY index, such as whether it's expected to go up or down. It could include the magnitude of the predicted movement, giving an idea of the potential

gain or loss.Details about when the algorithm recommends entering a trade, such as the specific date and time. Information on the exit strategy, including suggested stop-loss and profit-taking levels.The code may specify the type of trading instrument to use (e.g., call options, put options) based on the predicted market movement. The quantity or lot size for the recommended trade could be provided.Displaying the time or date at which the analysis was performed. Considerations or adjustments based on current market conditions.Information about risk management strategies, such as how much capital to allocate for the trade. Any additional risk measures or considerations.If options trading is involved, the code might provide insights from the options chain analysis, such as the strike price selected for trading.Any specific instructions or recommendations for the user or developer regarding the execution of trades or further actions.

3 Code

The script starts by importing the necessary libraries, including ‘requests‘, ‘tvDatafeed‘, ‘pandas‘, ‘talib‘, ‘SimpleImputer‘, and ‘datetime‘. It then reads a CSV file containing stock market data using the ‘pd.read_csv()‘ function and preprocesses the data by converting the ‘datetime‘ column to a datetime object using the ‘pd.to_datetime()‘ function. The script then engineers new features such as moving averages, MACD, and various candlestick patterns using the ‘talib‘ library.

The script then trains a random forest regression model using the preprocessed data for making predictions. It fetches real-time stock market data using the ‘tvDatafeed‘ library and engineers new features such as moving averages, MACD, and various candlestick patterns using the ‘talib‘ library. The script then uses the trained model to make predictions on the real-time data and makes trading decisions based on the predictions.

The script also fetches options chain data from the NSE website and makes trading decisions based on the options data. It prints the trading decisions, including the predicted price movement, entry time, stop loss, profit booking, and the options trading decisions.

The script is quite extensive and performs multiple tasks related to stock market data analysis and trading. It involves data preprocessing, machine learning model training, real-time data fetching, and trading decision making based on the predictions and options data. The script can be used as a part of an automated trading system that uses machine learning models for making trading decisions and options trading based on real-time data.

The script reads the CSV file containing stock market data using the ‘pd.read_csv()‘ function and preprocesses the data by converting the ‘datetime‘ column to a datetime object using the ‘pd.to_datetime()‘ function. The script then engineers new features such as moving averages, MACD, and various candlestick patterns using the ‘talib‘ library. The ‘talib‘ library is a technical analysis library that provides a wide range of technical indicators for financial analysis.

The script trains a random forest regression model using the preprocessed data for making predictions. The random forest regression model is a machine learning algorithm that is used for regression tasks. It works by constructing multiple decision trees and combining their predictions to obtain a final prediction. The random forest algorithm is known for its high accuracy and robustness to noise and outliers.

The script fetches real-time stock market data using the ‘tvDatafeed‘ library. The ‘tvDatafeed‘ library is a Python wrapper for the TradingView API, which provides real-time and historical market data for various financial instruments. The script engineers new features such as moving averages, MACD, and various candlestick patterns using the ‘talib‘ library. The script then uses the trained model to make predictions on the real-time data and makes trading decisions based on the predictions.

The script also fetches options chain data from the NSE website and makes trading decisions based on the options data. The options chain data provides information about the available options contracts for a particular stock or index, including the strike price, expiration date, and option type (call or put). The

script uses the options chain data to make trading decisions based on the predicted price movement.
The script prints the trading decisions, including the predicted price movement, entry time stop loss profit booking and options trading decisions. The script uses various Python libraries such as ‘pandas‘ ‘numpy‘ and ‘sklearn‘ for data manipulation, machine learning, and data visualization.] The script is a program that appears to be related to stock market data analysis and trading.The script performs several tasks, including data preprocessing, feature engineering, machine learning model training, real-time data fetching, and trading decision making based on the predictions and options data.
The script starts by importing the necessary libraries, including ‘requests‘, ‘tvDatafeed‘, ‘pandas‘, ‘talib‘, ‘SimpleImputer‘, and ‘datetime‘. It then reads a CSV file containing stock market data using the ‘pd.read_csv()‘ function and preprocesses the data by converting the ‘datetime‘ column to a datetime object using the ‘pd.to_datetime()‘ function. The script then engineers new features such as moving averages, MACD, and various candlestick patterns using the ‘talib‘ library.
The script then trains a random forest regression model using the preprocessed data for making predictions. It fetches real-time stock market data using the ‘tvDatafeed‘ library and engineers new features such as moving averages, MACD, and various candlestick patterns using the ‘talib‘ library. The script then uses the trained model to make predictions on the real-time data and makes trading decisions based on the predictions.
The script also fetches options chain data from the NSE website and makes trading decisions based on the options data. It prints the trading decisions, including the predicted price movement, entry time, stop loss, profit booking, and the options trading decisions.
The script is quite extensive and performs multiple tasks related to stock market data analysis and trading. It involves data preprocessing, machine learning model training, real-time data fetching, and trading decision making based on the predictions and options data. The script can be used as a part of an automated trading system that uses machine learning models for making trading decisions and options trading based on real-time data.
The script reads the CSV file containing stock market data using the ‘pd.read_csv()‘ function and preprocesses the data by converting the ‘datetime‘ column to a datetime object using the ‘pd.to_datetime()‘ function. The script then engineers new features such as moving averages, MACD, and various candlestick patterns using the ‘talib‘ library. The ‘talib‘ library is a technical analysis library that provides a wide range of technical indicators for financial analysis.
The script trains a random forest regression model using the preprocessed data for making predictions. The random forest regression model is a machine learning algorithm that is used for regression tasks. It works by constructing multiple decision trees and combining their predictions to obtain a final prediction. The random forest algorithm is known for its high accuracy and robustness to noise and outliers.
The script fetches real-time stock market data using the ‘tvDatafeed‘ library. The ‘tvDatafeed‘ library is a Python wrapper for the TradingView API, which provides real-time and historical market data for various

financial instruments. The script engineers new features such as moving averages, MACD, and various candlestick patterns using the ‘talib‘ library. The script then uses the trained model to make predictions on the real-time data and makes trading decisions based on the predictions.

The script also fetches options chain data from the NSE website and makes trading decisions based on the options data. The options chain data provides information about the available options contracts for a particular stock or index, including the strike price, expiration date, and option type (call or put). The script uses the options chain data to make trading decisions based on the predicted price movement.

The script prints the trading decisions, including the predicted price movement, entry time stop loss profit booking and options trading decisions. The script uses various Python libraries such as ‘pandas‘ ‘numpy‘ and ‘sklearn‘ for data manipulation, machine learning, and data visualization.

```
import requests
from tvDatafeed import TvDatafeed,Interval
import pandas as pd
import talib
from sklearn.impute import SimpleImputer
from datetime import datetime, timedelta
imputer = SimpleImputer(strategy='mean')
import time



path = pd.read_csv('NIFTY.csv')
five_min_data= pd.DataFrame(path)

five_min_data['datetime']=pd.to_datetime(five_min_data['datetime'])
five_min_data['year'] = five_min_data['datetime'].dt.year
five_min_data['hour'] = five_min_data['datetime'].dt.hour
five_min_data['minute'] = five_min_data['datetime'].dt.minute

#candlestick Patterns, macd, ema_14_period

five_min_data['EMA_14'] = talib.EMA(five_min_data['close'],timeperiod=14)
macd, macdsignal, macdhist = talib.MACD(five_min_data['close'], fastperiod
    =12, slowperiod=26, signalperiod=9)
five_min_data['MACD'] = macd
five_min_data['engulfing'] = talib.CDLENGULFING(five_min_data['open'],
    five_min_data['high'], five_min_data['low'],five_min_data['close'])
```

```
#five_min_data['dragonfly'] = talib.CDLDRAGONFLYDOJI(five_min_data['open'
    ], five_min_data['high'], five_min_data['low'],five_min_data['close'])
five_min_data['hammer'] = talib.CDLHAMMER(five_min_data['open'],
    five_min_data['high'], five_min_data['low'],five_min_data['close'])
#five_min_data['gravestone_doji'] = talib.CDLGRAVESTONEDOJI(five_min_data[
    'open'], five_min_data['high'], five_min_data['low'],five_min_data['
    close'])
five_min_data['shooting_star'] = talib.CDLSHOOTINGSTAR(five_min_data['open
    '], five_min_data['high'], five_min_data['low'],five_min_data['close'])
print(five_min_data)
five_min_data.columns

five_min_data=five_min_data.iloc[201:].reset_index(drop=True)
start_ind=[]
X_train=[]
y_train=[]
y_train_pos=[]
gain=[]

for j in range(len(five_min_data.index)):
    if five_min_data['hour'][j]==9 and five_min_data['minute'][j]==15: #
        Getting the indices where the trading day starts
         start_ind.append(j)
end_ind=[] # end of the day indices
for j in range(1,len(start_ind)): #starting from 1 because we dont want an
    end without the start of a day which is the case for start_of_day[0]-1
    end_ind.append(start_ind[j]-1)


if len(start_ind)>len(end_ind):
    start_ind=start_ind[:-1]

j=0
while j<len(start_ind):
    if end_ind[j]-start_ind[j]!=74:
        end_ind.remove(end_ind[j])
```

```
            start_ind.remove(start_ind[j])
            continue
        df_new=five_min_data.iloc[start_ind[j]+2:end_ind[j]-2].drop(['symbol',
            'datetime','year','hour','minute'],axis=1)

        #-------------------------------------------------------------------------------

        df_new = pd.DataFrame(imputer.fit_transform(df_new))
        if len(df_new.columns)==len(five_min_data.columns)-5:
            X_train.append(df_new.values.flatten())
        else:
            end_ind.remove(end_ind[j])
            start_ind.remove(start_ind[j])
        j+=1

for j in end_ind:
    gain_inst=(five_min_data['close'][j+7]-five_min_data['close'][j-2])
        #*100/five_min_data['close'][j-2] # gain at the particular instant
        diff. between 10:00am - 11:30am
    gain.append(gain_inst)  # i+2 indicates the next day at 9:20(as we
        need to let the price settle) and i-2 at time 3:15 as we need to
        trade before the day ends and we cant trade ecaxtly at the end of
        the day

y_train=gain


from sklearn.preprocessing import MinMaxScaler

# Normalizing the data
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)


from sklearn.ensemble import RandomForestRegressor
from sklearn.utils import shuffle
```

```
#shuffled data
X_train_shuffled, y_train_shuffled = shuffle(X_train, y_train,
    random_state=42)
rf_regressor=RandomForestRegressor(n_estimators=60,random_state=42)
rf_regressor.fit(X_train_shuffled,y_train_shuffled)


tv=TvDatafeed()

df = tv.get_hist(symbol='NIFTY',exchange='NSE',interval=Interval.
    in_5_minute,n_bars=750)
df_n= pd.DataFrame(df)
df_n=df_n.reset_index()
df_n['datetime']=pd.to_datetime(df_n['datetime'])
df_n['year'] = df_n['datetime'].dt.year
df_n['hour'] = df_n['datetime'].dt.hour
df_n['minute'] = df_n['datetime'].dt.minute
#-------------------------------------------------------------------------------------


df_n['EMA_14'] = talib.EMA(df_n['close'],timeperiod=14)
macd, macdsignal, macdhist = talib.MACD(df_n['close'], fastperiod=12,
    slowperiod=26, signalperiod=9)
df_n['MACD'] = macd
df_n['engulfing'] = talib.CDLENGULFING(df_n['open'], df_n['high'], df_n['
    low'],df_n['close'])
#df_n['dragonfly'] = talib.CDLDRAGONFLYDOJI(df_n['open'], df_n['high'],
    df_n['low'],df_n['close'])
df_n['hammer'] = talib.CDLHAMMER(df_n['open'], df_n['high'], df_n['low'],
    df_n['close'])
#df_n['gravestone_doji'] = talib.CDLGRAVESTONEDOJI(df_n['open'], df_n['
    high'], df_n['low'],df_n['close'])
df_n['shooting_star'] = talib.CDLSHOOTINGSTAR(df_n['open'], df_n['high'],
    df_n['low'],df_n['close'])
#df_n.to_csv('test1.csv')
```

```
df_n= df_n.iloc[201:].reset_index(drop=True)
df_n.to_csv('test2.csv')
#-------------------------------------------------------------------------------------

engulf=[]
engulf_sell=[0]

for i in df_n.index:
    if df_n['engulfing'][i] >0 :
        engulf.append(i)


y = len(df_n.index)-engulf[-1]

x_pred=[]
price_arr=[]

df_n=df_n.reset_index(drop=True)
df_nn=df_n.iloc[-69-y:-y+1].drop(['symbol','datetime','year','hour','
   minute'],axis=1)
#df_nn.to_csv('trading data.csv')
x_pred.append(df_nn.values.flatten())
x_pred_norm=scaler.transform(x_pred)
y_pred=rf_regressor.predict(x_pred)
z=df_n['datetime'][engulf[-1]]
print(f"\nPrdecting Nifty to go {y_pred} points from {z} NIFTY at {df_n['
   close'][engulf[-1]]}")
print(f"\nEntry time :- {z}\nStop Loss at: {.3*y_pred} \nProfit book at :
   {.9*y_pred}")


##-------------------------------------------------------------------------------------


url = 'https://www.nseindia.com/api/option-chain-indices?symbol=NIFTY'
```

```
headers= {"Accept-Encoding" : "gzip, deflate, br",
          "Accept-Language": "en-GB,en-US;q=0.9,en;q=0.8",
          "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
              AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0
              Safari/537.36"}

session = requests.Session()
data = session.get(url, headers=headers).json()['records']['data']
ocdata =[]
for i in data:
    for j,k in i.items():
        if j=='CE' or j=="PE":
            info=k
            info["instrumentType"]= j
            ocdata.append(info)

ddf = pd.DataFrame(ocdata).drop(['underlying', 'openInterest','
   changeinOpenInterest', 'pchangeinOpenInterest', 'totalTradedVolume','
   impliedVolatility', 'change', 'pChange','totalBuyQuantity', '
   totalSellQuantity', 'bidQty', 'bidprice', 'askQty','askPrice', '
   underlyingValue', 'instrumentType'],axis=1)

subt = (df_n['close'][engulf[-1]])%50
st_price=(int(df_n['close'][engulf[-1]]-subt))
options = []
for i in ddf.index:
    if ddf['strikePrice'][i] == st_price:
        options.append(i)

intiatl_bal = 25000
call_position=[]
sell_position=[]
for i in options[0:2]:
    if y_pred>0 and "CE" in ddf['identifier'][i]:
        call_posn = int(intiatl_bal/ddf['lastPrice'][i])
        call_posn = int(call_posn/50)
```

```
        print(f"\nBuy {call_posn} lot of CE strike price: -- {ddf['
            strikePrice'][i]}")

    elif y_pred<0 and "PE" in ddf['identifier'][i]:
        call_posn = int(intiatl_bal/ddf['lastPrice'][i])
        call_posn = int(call_posn/50)
        print(f"\n Buy {call_posn} lot of PE strike price: -- {ddf['
            strikePrice'][i]}")
```

4 Testing the Algorithm

1. **Trade 1: Predicted on 5-01-2024**

```
Prdecting Nifty to go [47.57] points from 2024-01-05 13:00:00 NIFTY at 21690.75

Entry time :- 2024-01-05 13:00:00
Stop Loss at: [14.271]
Profit book at : [42.813]

Buy 1 lot of CE strike price: -- 21650
```

Figure 1: Prdeicted Trade on 5-01-2024

Figure 2: Result for trade 1, Stop loss trade

2. **Trade 2: Predicted on 8-01-2024**

```
Prdecting Nifty to go [30.77333333] points from 2024-01-08 13:55:00 NIFTY at 21585.25

Entry time :- 2024-01-08 13:55:00
Stop Loss at: [9.232]
Profit book at : [27.696]

Buy 1 lot of CE strike price: -- 21550
```

Figure 3: Prdeicted Trade on 8-01-2024

Figure 4: Result for trade 2, Stop loss trade

3. **Trade 3: Predicted on 9-01-2024**

```
Prdecting Nifty to go [57.40416667] points from 2024-01-09 09:55:00 NIFTY at 21625.75

Entry time :- 2024-01-09 09:55:00
Stop Loss at: [17.22125]
Profit book at : [51.66375]

Buy 1 lot of CE strike price: -- 21600
```

Figure 5: Prdeicted Trade on 9-01-2024

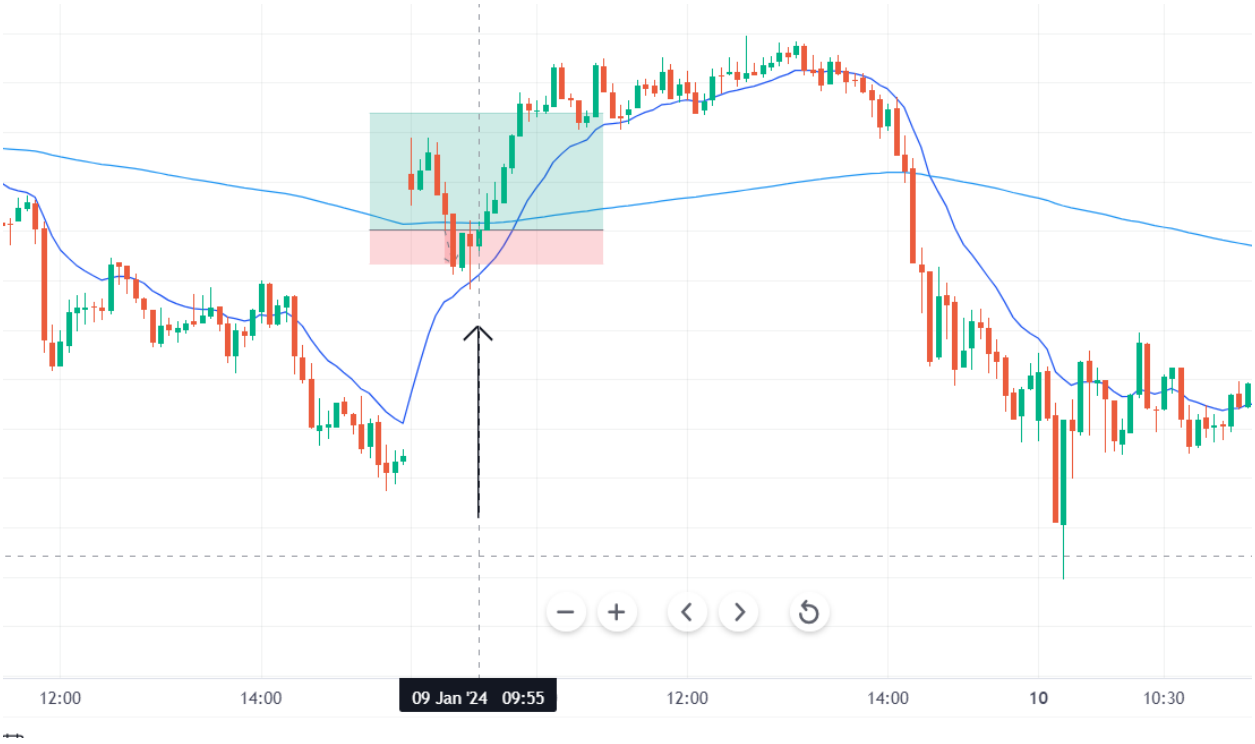

Figure 6: Result for trade 3, Target Achieved

4. **Trade 4: Predicted on 11-01-2024**

```
Prdecting Nifty to go [51.6375] points from 2024-01-11 10:55:00 NIFTY at 21657.8

Entry time :- 2024-01-11 10:55:00
Stop Loss at: [15.49125]
Profit book at : [46.47375]

Buy 1 lot of CE strike price: -- 21650
```

Figure 7: Prdeicted Trade on 11-01-2024

Figure 8: Result for trade 4, Stop Loss

5. **Trade 5: Predicted on 12-01-2024**

```
Prdecting Nifty to go [98.00833333] points from 2024-01-12 11:15:00 NIFTY at 21832.25

Entry time :- 2024-01-12 11:15:00
Stop Loss at: [29.4025]
Profit book at : [88.2075]

Buy 1 lot of CE strike price: -- 21800
```

Figure 9: Prdeicted Trade on 12-01-2024

Figure 10: Result for trade 5, Target Achieved

6. **Trade 6: Predicted on 12-01-2024**

```
Prdecting Nifty to go [104.8725] points from 2024-01-15 10:35:00 NIFTY at 22031.3

Entry time :- 2024-01-15 10:35:00
Stop Loss at: [31.46175]
Profit book at : [94.38525]

Buy 3 lot of CE strike price: -- 22000
```

Figure 11: Prdeicted Trade on 15-01-2024

Figure 12: Result for trade 6, Target Achieved

5 Real Market Live Trade

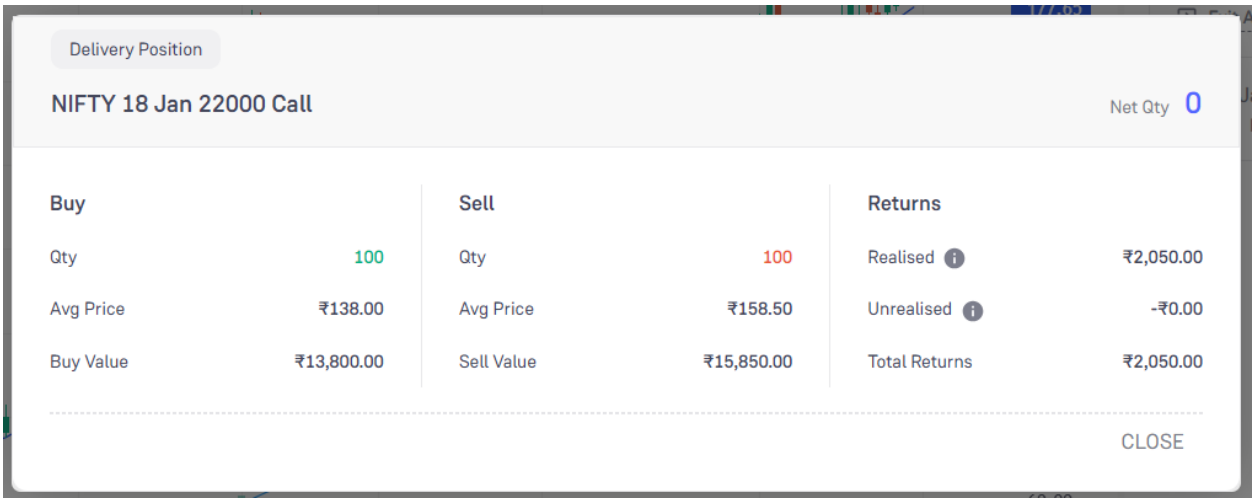

Figure 13: Position Details

Figure 14: Result for trade 6, Target Achieved

During the 6th trade of the testing phase, a live trade was executed based on the model's prediction. Utilizing a capital of 13,800 rupees, two lots of NIFTY 22000 strike price call options were acquired at a rate of 138 rupees each. The position was later closed when the option price reached 158 rupees.

6 Conclusion

A total of 6 trades were executed based on fresh data, with successful outcomes in three instances, resulting in a prediction accuracy of 50%. Despite the moderate accuracy, the model proves to be profitable due to a favorable risk-reward ratio, where each profitable trade yields three times the profit compared to a single loss. Consequently, one profitable trade compensates for three unsuccessful ones, leaving room for overall profitability. This model is specifically designed for purchasing call options, and a similar approach could be applied to creating a model focused on option selling. Further enhancements in accuracy can be achieved by incorporating more sophisticated models, additional indicators, and the analysis of candlestick patterns.