Centre for Development of Advanced Computing, Mumbai

Report on

**Algorithmic trading for Financial markets**

PG-DBDA March 2022

Submitted by:

Project Team 06

**Swapni Adhav Abhishek Raghorte**

**Vishvnath Hambire Rahul Dubal**

**Pratik Satham Priyanka Bharshankar**

**Mr Abhishek Jain**

**1 INTRODUCTION**

Algorithmic trading (also called automated trading, black-box trading, or algo-trading) uses a computer program that follows a defined set of instructions (an algorithm) to place a trade. The trade, in theory, can generate profits at a speed and frequency that is impossible for a human trader.

The defined sets of instructions are based on timing, price, quantity, or any mathematical model. Apart from profit opportunities for the trader, algo-trading renders markets more liquid and trading more systematic by ruling out the impact of human emotions on trading activities.

The user will be able to select a strategy and if the condition of the strategy are met then the stocks will be automatically bought/sold. We have also considered to use of stoploss if the stocks reversed the trend. We have set 2% of buy price as our target price and 1% of buy price as our stoploss price. At maximum we can chose as much as 50 stocks to trade. The process is active during market hours only.

We have used FYERS API integration which will help us execute the buy/sell instructions from our trading strategies in real-time hence, resulting in better and more accurate order execution.

## **2. PROBLEM STATEMENT**

Most new traders tend to start with the traditional form of trading wherein they analyse the market by themselves, identify potential trading opportunities, and manually place their trades. If the investor is not trained or doesn’t know the basics of stock market. He will lose a lot of money. Untrained investor buys the stock based on some tips but do not consider the risk management. Some of them don’t put stoploss and end up losing a lot of money.

However, many would want to know **how good algo trading *is*** before they decide to give it a try. **Algo trading is good because it reduces the chances of human errors that are commonly seen in discretionary trading. Moreover, with algo trading, you also have better control of statistics since your trading edge and the odds even before launching your algo trading system.**

Strategies Used

1. Opening Range Breakout(ORB)

Opening range breakouts are one of the important reversal and continuation chart patterns,  designed to capture move or reversal during this first hour.

The first hour of the trading day is the most active and dynamic period. Though it is the time period where you can make most of your money quickly, you may also lose without a trading plan. The first hour is the most volatile time frame during the trading day.

The opening range is high and low for a given period after the market opens. This period is generally the first 30 or 60 minutes of trading. It is one most important chart patterns to make money in the stock market.

During this period we need to identify the highs and lows of the day. Also, we need to [identify pre-market highs and lows](https://blog.stockedge.com/understanding-edge-reports-in-stockedge/?utm_campaign=blog_CTA&utm_medium=blogpage&utm_source=elearnmarkets_blog), as these levels act like a magnet on price action after the market opens.

The opening hour of the market is associated with big trading volumes and volatility. This time of the trading session provides many trading opportunities. In this way, traders use the opening range to set the entry points and to predict and forecast the price action for the day.

ORB Strategy

# pip install fyers-apiv2  
# pip install selenium  
# pip install webdriver-manager  
  
from fyers\_api.Websocket import ws  
from fyers\_api import fyersModel  
from fyers\_api import accessToken  
import datetime  
import time  
import document\_file  
from selenium import webdriver  
from webdriver\_manager.firefox import GeckoDriverManager  
  
log\_path = document\_file.log\_path  
client\_id = document\_file.client\_id  
secret\_key = document\_file.secret\_key  
redirect\_url = document\_file.redirect\_url  
response\_type = document\_file.response\_type  
grant\_type = document\_file.grant\_type  
username = document\_file.username  
password = document\_file.password  
pin1 = document\_file.pin1  
pin2 = document\_file.pin2  
pin3 = document\_file.pin3  
pin4 = document\_file.pin4  
  
client\_id = '8S56FEB6JI-100'  
secret\_key = 'BGOWTLDD3I'  
redirect\_url = 'http://127.0.0.1:5000/login'  
  
open\_position = []  
  
  
def getTime():  
 return datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')  
  
  
def custom\_message(msg):  
 # print(msg)  
 script = msg[0]['symbol']  
 ltp = msg[0]['ltp']  
 high = msg[0]['high\_price']  
 low = msg[0]['low\_price']  
 ltt = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(msg[0]['timestamp']))  
 print(f"Script: {script}, Ltp:{ltp}, High:{high}, Low:{low}, ltt:{ltt}")  
  
 if (ltp <= (low)) and (script not in open\_position):  
 open\_position.append(script)  
 placeOrder("SELL", script, ltp)  
  
 if (ltp >= (high)) and (script not in open\_position):  
 open\_position.append(script)  
 placeOrder("BUY", script, ltp)  
  
  
def placeOrder(order, script, ltp):  
 if order == "BUY":  
 quantity = int(1)  
 target\_price = int(ltp \* 0.02)  
 stoploss\_price = int(ltp \* 0.01)  
  
 order = fyers.place\_order(  
 {"symbol": script, "qty": quantity, "type": "2", "side": "1", "productType": "BO", "limitPrice": "0",  
 "stopPrice": "0", "disclosedQty": "0", "validity": "DAY", "offlineOrder": "False",  
 "stopLoss": stoploss\_price, "takeProfit": target\_price})  
 print(  
 f"Buy Order Placed for {script}, at Price: {ltp} for Quantity: {quantity}, with order\_id: {order['id']} at time: {getTime()}")  
 print(open\_position)  
  
 else:  
 quantity = int(1)  
 target\_price = int(ltp \* 0.02)  
 stoploss\_price = int(ltp \* 0.01)  
  
 order = fyers.place\_order(  
 {"symbol": script, "qty": quantity, "type": "2", "side": "-1", "productType": "BO", "limitPrice": "0",  
 "stopPrice": "0", "disclosedQty": "0", "validity": "DAY", "offlineOrder": "False",  
 "stopLoss": stoploss\_price, "takeProfit": target\_price})  
 print(  
 f"Sell Order Placed for {script}, at Price: {ltp} for Quantity: {quantity}, with order\_id: {order['id']} at time: {getTime()}")  
 print(open\_position)  
  
  
def generate\_access\_token(auth\_code, client\_id, secret\_key):  
 appSession = accessToken.SessionModel(client\_id=client\_id, secret\_key=secret\_key, grant\_type="authorization\_code")  
 appSession.set\_token(auth\_code)  
 response = appSession.generate\_token()["access\_token"]  
 return response  
  
  
def generate\_auth\_code():  
 session = accessToken.SessionModel(client\_id=client\_id, secret\_key=secret\_key, redirect\_uri=redirect\_url,  
 response\_type='code', grant\_type='authorization\_code')  
 response = session.generate\_authcode()  
 print("login URL", response)  
 auth\_code = input("Enter auth code :")  
 return auth\_code  
  
  
def main():  
 global fyers  
  
 auth\_code = generate\_auth\_code()  
 access\_token = generate\_access\_token(auth\_code, client\_id, secret\_key)  
 fyers = fyersModel.FyersModel(token=access\_token, log\_path=log\_path, client\_id=client\_id)  
 fyers.token = access\_token  
 newtoken = f"{client\_id}:{access\_token}"  
 data\_type = "symbolData"  
  
 symbol = ["NSE:ICICIPRULI-EQ", "NSE:GLENMARK-EQ", "NSE:WIPRO-EQ", "NSE:SYNGENE-EQ", "NSE:DLF-EQ"]  
  
 # symbol = ["MCX:CRUDEOIL22MARFUT", "MCX:GOLDM22MARFUT"]  
  
 orderplacetime = int(9) \* 60 + int(20)  
 closingtime = int(13) \* 60 + int(35)  
 timenow = (datetime.datetime.now().hour \* 60 + datetime.datetime.now().minute)  
 print(f"Waiting for 9.20 AM , Time Now:{getTime()}")  
  
 while timenow < orderplacetime:  
 time.sleep(0.2)  
 timenow = (datetime.datetime.now().hour \* 60 + datetime.datetime.now().minute)  
 print(f"Ready for trading, Time Now:{getTime()}")  
  
 fs = ws.FyersSocket(access\_token=newtoken, run\_background=False, log\_path=log\_path)  
 fs.websocket\_data = custom\_message  
 fs.subscribe(symbol=symbol, data\_type=data\_type)  
 fs.keep\_running()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

1. Relative Strength Indicator (RSI Strategy)

The relative strength index (RSI) is a momentum indicator used in technical analysis. RSI measures the speed and magnitude of a security's recent price changes to evaluate overvalued or undervalued conditions in the price of that security.

The RSI is displayed as an oscillator (a line graph) on a scale of zero to 100.

The RSI can do more than point to overbought and oversold securities. It can also indicate securities that may be primed for a trend reversal or corrective pullback in price. It can signal when to buy and sell. Traditionally, an RSI reading of 70 or above indicates an overbought situation. A reading of 30 or below indicates an oversold condition.

# pip install fyers-apiv2  
# pip install selenium  
# pip install webdriver-manager  
# pip install pandas  
# pip install TA\_Lib-0.4.24-cp310-cp310-win\_amd64.whl  
# https://www.lfd.uci.edu/~gohlke/pythonlibs/#ta-lib  
import os  
  
from fyers\_api.Websocket import ws  
from fyers\_api import fyersModel  
from fyers\_api import accessToken  
import datetime  
import time  
import document\_file  
from selenium import webdriver  
from webdriver\_manager.firefox import GeckoDriverManager  
import talib as ta  
import pandas as pd  
  
#from final import generate\_auth\_code, generate\_access\_token, log\_path  
  
client\_id = '8S56FEB6JI-100'  
secret\_key = 'BGOWTLDD3I'  
redirect\_url = 'http://127.0.0.1:5000/login'  
log\_path = os.getcwd()  
  
  
script\_list = ["HDFCBANK-EQ","SBIN-EQ","INFY-EQ","ICICIBANK-EQ","AXISBANK-EQ","MARUTI-EQ","WIPRO-EQ","BHARTIARTL-EQ","ASIANPAINT-EQ","DIVISLAB-EQ","HDFC-EQ"]  
exchange = "NSE"  
quantity = int(1)  
timeframe = "1"  
from\_date = "2022-07-10"  
today = datetime.datetime.now().strftime('%Y-%m-%d') #"2022-09-29"  
rsi\_overbought = 80  
rsi\_oversold = 20  
buy\_traded\_stock = []  
sell\_traded\_stock = []  
  
def getTime():  
 return datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')  
  
def placeOrder(script, order):  
 if order == "BUY":  
 order = fyers.place\_order({"symbol":f"{exchange}:{script}","qty":quantity,"type":"2","side":"1","productType":"INTRADAY","limitPrice":"0","stopPrice":"0","disclosedQty":"0","validity":"DAY","offlineOrder":"False","stopLoss":"0","takeProfit":"0"})  
 print(f"Buy Order Placed for {script} at time: {getTime()}")  
 else:  
 order = fyers.place\_order({"symbol":f"{exchange}:{script}","qty":quantity,"type":"2","side":"-1","productType":"INTRADAY","limitPrice":"0","stopPrice":"0","disclosedQty":"0","validity":"DAY","offlineOrder":"False","stopLoss":"0","takeProfit":"0"})  
 print(f"Buy Order Placed for {script} at time: {getTime()}")  
  
def rsiAlgorithm():  
 for script in script\_list:  
 data = {"symbol":f"{exchange}:{script}","resolution": timeframe,"date\_format":"1","range\_from": from\_date,"range\_to": today,"cont\_flag":"0"}  
 try:  
 hist\_data = fyers.history(data)  
  
 except Exception as e:  
 raise e  
 hist\_data = hist\_data['candles']  
 df=pd.DataFrame(hist\_data, columns=['date', 'open', 'high', 'low', 'close', 'volume'])  
  
 df['date'] = pd.to\_datetime(df['date'], unit = "s", utc=True)  
 df['date'] = df['date'].dt.tz\_convert('Asia/Kolkata')  
 df["rsi"] = ta.RSI(df["close"], timeperiod=14).round(2)  
  
 df.dropna(inplace=True)  
 if not df.empty:  
 print(df)  
 rsi\_value = df.rsi.values[-1]  
 # print(df)  
 if (rsi\_value >= rsi\_overbought) and (script not in sell\_traded\_stock):  
 sell\_traded\_stock.append(script)  
 placeOrder(script, "SELL")  
  
 if (rsi\_value <= rsi\_oversold) and (script not in buy\_traded\_stock):  
 buy\_traded\_stock.append(script)  
 placeOrder(script, "BUY")  
  
def generate\_access\_token(auth\_code, client\_id, secret\_key):  
 appSession = accessToken.SessionModel(client\_id=client\_id, secret\_key=secret\_key, grant\_type="authorization\_code")  
 appSession.set\_token(auth\_code)  
 response = appSession.generate\_token()["access\_token"]  
 return response  
  
  
def generate\_auth\_code():  
 session = accessToken.SessionModel(client\_id=client\_id, secret\_key=secret\_key, redirect\_uri=redirect\_url,  
 response\_type='code', grant\_type='authorization\_code')  
 response = session.generate\_authcode()  
 print("login URL", response)  
 auth\_code = input("Enter auth code :")  
 return auth\_code  
  
  
def main():  
 global fyers  
  
 auth\_code = generate\_auth\_code()  
 access\_token = generate\_access\_token(auth\_code, client\_id, secret\_key)  
 fyers = fyersModel.FyersModel(token=access\_token, log\_path=log\_path, client\_id=client\_id)  
 fyers.token = access\_token  
 newtoken = f"{client\_id}:{access\_token}"  
 data\_type = "symbolData"  
  
  
 orderplacetime = int(9) \* 60 + int(20)  
 closingtime = int(13) \* 60 + int(35)  
 print("close",closingtime)  
 timenow = (datetime.datetime.now().hour \* 60 + datetime.datetime.now().minute)  
 print(f"Waiting for 9.20 AM , Time Now:{getTime()}")  
  
 while timenow < orderplacetime:  
 time.sleep(0.2)  
 timenow = (datetime.datetime.now().hour \* 60 + datetime.datetime.now().minute)  
 print(f"Ready for trading, Time Now:{getTime()}")  
  
  
  
  
  
 while timenow < closingtime:  
 rsiAlgorithm()  
  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

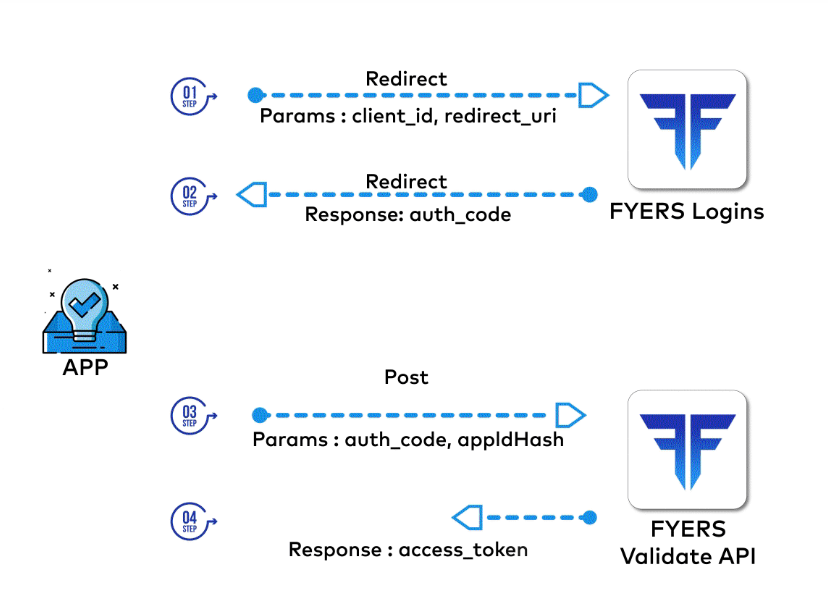
FYERS API

Fyers API is a set of **REST-like APIs** that provide integration with our in-housetrading platform with which we can build your own customized trading applications.

Fyers API is a set of **REST-like APIs** that provide integration with our in-house trading platform with which you can build your own customized trading applications. You can place fresh single or multiple orders, modify and cancel existing orders in real-time. You can also get account related information such as orderbook, tradebook, net positions, holdings, and funds.

We have ensured maximum security for our APIs which prevent unauthorised transactions. All API requests and received only over HTTPS protocol.

For more information about FYERS APIs [here](https://fyers.in/introducing-trading-apis/)



APi code

from fyers\_api import fyersModel  
from fyers\_api import accessToken  
import os  
  
client\_id = '8S56FEB6JI-100'  
secret\_key = 'BGOWTLDD3I'  
redirect\_url = 'http://127.0.0.1:5000/login'  
  
  
def get\_access\_token():  
 if not os.path.exists('access\_token.txt'):  
 session = accessToken.SessionModel(client\_id=client\_id, secret\_key=secret\_key, redirect\_uri=redirect\_url,  
 response\_type='code', grant\_type='authorization\_code')  
 response = session.generate\_authcode()  
 print("login URL", response)  
 auth\_code = input("Enter auth code :")  
 session.set\_token(auth\_code)  
 access\_token = session.generate\_token()['access\_token']  
 with open('access\_token.txt', 'w') as f:  
 f.write(access\_token)  
  
 else:  
 with open('access\_token.txt', 'r') as f:  
 access\_token = f.read()  
 return access\_token  
  
  
  
  
  
  
fyers = fyersModel.FyersModel(client\_id=client\_id, token=get\_access\_token(), log\_path="D:\projects")  
  
print(fyers.get\_profile())  
  
data = {"symbol":"NSE:SBIN-EQ","resolution":"D","date\_format":"1","range\_from":"2022-09-01","range\_to":"2022-09-10","cont\_flag":"1"}  
is\_async = True  
print(fyers.history(data))