In [3]:

```
import numpy as np # for numerical computing
import pandas as pd # for tabular data manipulation
import matplotlib.pyplot as plt # for data graphics

from datetime import datetime
from dateutil.relativedelta import relativedelta

%matplotlib inline

import warnings
# To ignore all warnings
warnings.filterwarnings("ignore")
import os
import glob

from IPython.display import display
from IPython.display import Image


import time
from time import sleep
import logging
import requests

from urllib.parse import urlencode
import hmac
import hashlib

from sklearn.linear_model import Lasso

from dotenv import load_dotenv



#### a
logging.basicConfig(format='%(asctime)s [%(threadName)-12.12s] [%(levelname)-5.5s]
                    level=logging.INFO)

# Base URLs
BASE_URL = 'https://testnet.binancefuture.com'


# Tier represent a price level
class Tier:
    def __init__(self, price: float, size: float, quote_id: str = None):
        self.price = price
        self.size = size
        self.quote_id = quote_id


# OderBook class to hold bids and asks, as an array of Tiers
class OrderBook:
    def __init__(self, _timestamp: float, _bids: [Tier], _asks: [Tier]):
        self.timestamp = _timestamp
        self.bids = _bids
        self.asks = _asks

    # method to get best bid
    def best_bid(self):
#          print(len(self.bids))
        # tested the bids lenth is 500
        return self.bids[0].price
    # method to get best ask
    def best_ask(self):
#         print(len(self.asks))
        # tested the asks lenth is 500
        return self.asks[0].price

    def mid(self):
        return 0.5 * (self.best_ask() + self.best_bid())

    def spread(self):
        return self.best_ask() - self.best_bid()


    def n_bid_ask(self, n: int):
        n_orders = []
        for i in range(n):
            n_orders +=\
            [
                self.bids[i].price,
                self.bids[i].size,
                self.asks[i].price,
                self.asks[i].size,
            ]

        return n_orders


# define a method that parse json message to order book object
def parse(json_object: {}) -> OrderBook:
    # process bids side
    bids = []
    for level in json_object['bids']:
        _price = float(level[0])
        _size = float(level[1])
        tier = Tier(_price, _size)
        bids.append(tier)

    # process asks side
    asks = []
    for level in json_object['asks']:
        _price = float(level[0])
        _size = float(level[1])
        tier = Tier(_price, _size)
        asks.append(tier)

    # "T" or "Trade time" is the time of the transaction in milliseconds, divide by
    _event_time = float(json_object['T']) / 1000
    return OrderBook(_event_time, bids, asks)


#### b
def collect_data_1s(batchN=200, n_orders=20, train_hr=1):
#     while True:
    n_file = 1
    log_list = []
    itr = 0
    data_entry = 0

    present =\
        datetime\
        .today()\
        .strftime("%Y%m%d")
    while itr < train_hr * 3600:
        data_entry +=1
        itr += 1
        # Get request to get order book snapshot of BTCUSDT
        response = requests.get('https://fapi.binance.com/fapi/v1/depth', params={'s

        # Get json object from response
        json_object = response.json()

        # call parse() method to convert json message to order book object
        order_book = parse(json_object)

        # print top of book
#         logging.info('[{}] = {}, {}'.format(datetime.fromtimestamp(order_book.times
        log_list +=\
        [
            [datetime.fromtimestamp(order_book.timestamp),
             order_book.mid(),
             order_book.spread(),]\
            + order_book.n_bid_ask(n_orders)
        ]
        if data_entry == batchN:
            temp_date =\
                datetime\
                .today()\
                .strftime("%Y%m%d")

            if temp_date != present:
                present = temp_date
                n_file = 1

            col_name =\
                ['time', 'mid_price', 'spread']
            for i in range(n_orders):
                col_name +=\
                [
                    f'bid{i}',
                    f'b_s{i}',
                    f'ask{i}',
                    f'a_s{i}'
                ]
            data_1s =\
                pd.DataFrame(log_list, columns = col_name)
            file_name = f'###BTCF1s_{present}_{n_file}.csv'
            data_1s.to_csv(file_name)

            print(file_name, " successfully logged")
            n_file += 1
            data_entry = 0
            log_list = []
        # sleep 1 second
        time.sleep(1)
    return

#### c
def get_credential():
    # Get the directory path of the current working directory
    script_directory = os.getcwd()
    # Define the relative path to the .env file
    relative_path = 'vault\mysecret.env'

    # Join the script directory with the relative path
    dotenv_path = os.path.join(script_directory, relative_path)

    # load_dotenv(dotenv_path=dotenv_path)
    result = load_dotenv(dotenv_path=dotenv_path)
    print("dotenv loaded successfully:", result)
    API_key = os.getenv('key')
    API_secret = os.getenv('secret')
    API_Docs = os.getenv('Docs')
    return API_key, API_secret


#### 1.2
def model_training(models_list, MLdata_df, train_pct, period,
                   y_idx, x_idx_list, threshold=0.5):
    Y = MLdata_df.iloc[:,[y_idx]].shift(-period).dropna()

    X = MLdata_df.iloc[:-period,x_idx_list]
    partition = int(len(MLdata_df)*train_pct)
    Y_train, Y_test = Y[:partition], Y[partition:]
    X_train, X_test = X[:partition], X[partition:]

#     print(Y_train.shape[0]==X_train.shape[0],
#           Y_test.shape[0]==X_test.shape[0])

    ML_res = []
    predicted = []
    names = []
    accuracy = []

    ML_mean =\
        Y_train.mean()
    ML_std =\
        Y_train.iloc[:,0].std()/\
        np.sqrt(9*3600*train_pct)

#     print(ML_mean, ML_std)
    win_max = 0

    for name, model in models_list:
        model_res = model.fit(X_train, Y_train)

        ML_pred = model_res.predict(X_test)

        acc =\
           MLdata_df\
            .iloc[:,[y_idx]]\
            .iloc[partition:-period]




        acc['pred'] = ML_pred

        acc['real'] = Y_test
        acc['pred_diff'] =\
            (acc.iloc[:,1]-acc.iloc[:,0])
        acc['real_diff'] =\
            (acc.iloc[:,2]-acc.iloc[:,0])

        mask_diff=acc['pred_diff'].copy()
        mask_diff[abs(mask_diff) <= threshold * ML_std]=0
        acc['mask_diff'] =\
            mask_diff
        acc['acc'] =\
            np.sign(
                acc['pred_diff'] * acc['real_diff']
            )
        acc['mask_acc'] =\
            np.sign(
                acc['mask_diff'] * acc['real_diff']

            )

        accuracy += [acc]
        wins = acc.iloc[:,-1].value_counts().loc[1]
        lose = acc.iloc[:,-1].value_counts().loc[-1]
        win_rate = wins/(wins+lose)
        if win_rate > win_max:
            win_max = win_rate
            win_name = name
        print(f"ML training for {name} done")
        print(f'number of sig: {wins+lose}, win rate is {win_rate*100}%')
        print(f"wins {wins}, loses {lose}")

        ML_res += [[name, acc, model_res]]


    print(f"most accurate model is {win_name} at {win_max*100}% accuracy")

    return ML_res

#### 1.3

def pnl_simulate(ML_model, data_1s, period=10,
                 y_idx=1, x_idx_list=np.arange(2,83),
                 TH=3, size=0.01):

    Y =\
        data_1s\
        .iloc[:,[y_idx]]\
        .shift(-period)\
        .dropna()

    X =\
        data_1s\
        .iloc[:-period,x_idx_list]

    y_pred =\
        ML_model.predict(X)
    pnl =\
        pd.concat([data_1s.iloc[:-period,:y_idx+1],
                   X.iloc[:,:9]],
                  axis=1)

    pnl['pred_diff'] =\
        y_pred - pnl['mid_price']

    pnl['position'] =\
        np.sign(pnl['pred_diff'])
    print(f'prediction threshold is {TH}')
    pnl.loc[abs(pnl['pred_diff']) <= TH, 'position'] = np.nan
    pnl['position'][0] = 0
    pnl['position'].ffill(inplace=True)
    pnl['position'].iloc[-1] = 0
    pnl['sig'] =\
        pnl['position'].diff()
    pnl['sig'][0] = 0
    pnl['pnl'] = 0
    pnl['profit'] = 0
    current_pos = 0
    for idx in pnl.index:
        sig = pnl.loc[idx, 'sig']
        if current_pos == 0:
            if sig > 0:
                current_pos =\
                    sig * size *\
                    pnl.loc[idx,'ask0']
            elif sig < 0:
                current_pos =\
                    sig * size *\
                    pnl.loc[idx,'bid0']

        elif current_pos > 0:
            value =\
                size *\
                pnl.loc[idx,'bid0']
            pnl.loc[idx, 'pnl'] =\
                value - current_pos
            if sig < 0:
                pnl.loc[idx, 'profit'] =\
                    pnl.loc[idx, 'pnl']
                if sig < -1:
                    current_pos = -value
        elif current_pos < 0:
            value =\
                -size *\
                pnl.loc[idx,'ask0']
            pnl.loc[idx, 'pnl'] =\
                value - current_pos
            if sig > 0:
                pnl.loc[idx, 'profit'] =\
                    pnl.loc[idx, 'pnl']
                if sig > 1:
                    current_pos = -value
    pnl['win'] =\
        np.sign(pnl['profit'])


    return pnl
#### 2.1
def recent_trade(key: str, secret: str, symbol: str):
    # market recent trade
    params = {
        "symbol": symbol,
    }

    # create query string
    query_string = urlencode(params)
    logging.info('Query string: {}'.format(query_string))

    # signature
    # signature
    signature = hmac.new(secret.encode("utf-8"), query_string.encode("utf-8"), hashl
    # url
    url = BASE_URL + '/fapi/v1/trades' + "?" + query_string + "&signature=" + signat

    # Define the request headers
    headers = {
        'X-MBX-APIKEY': key
    }

    # Send the request to get all open orders
    response = requests.get(url, headers=headers)
    return response.json()

#### 2.2
def send_market_order(key: str, secret: str, symbol: str, quantity: float, side: boo
    # order parameters
    timestamp = int(time.time() * 1000)
    params = {
        "symbol": symbol,
        "side": "BUY" if side else "SELL",
        "type": "MARKET",
        "quantity": quantity,
        'timestamp': timestamp - 500
    }

    # create query string
    query_string = urlencode(params)
    logging.info('Query string: {}'.format(query_string))

    # signature
    signature = hmac.new(secret.encode("utf-8"), query_string.encode("utf-8"), hashl

    # url
    url = BASE_URL + '/fapi/v1/order' + "?" + query_string + "&signature=" + signatu

    # post request
    session = requests.Session()
    session.headers.update(
        {"Content-Type": "application/json;charset=utf-8", "X-MBX-APIKEY": key}
    )
    response = session.post(url=url, params={})

    # get order id
    response_map = response.json()
    order_id = response_map.get('orderId')
    print(order_id)
    return order_id


"""

Create a method to send limit order

"""

def send_limit_order(key: str, secret: str, symbol: str, quantity: float, side: bool
    # order parameters
    timestamp = int(time.time() * 1000)


    # Order parameters
    params = {
        "symbol": symbol,
        "side": "BUY" if side else "SELL",
        "type": "LIMIT",
        "quantity": quantity,
        "price": price,
        "timeInForce": TIF,
        'timestamp': timestamp - 500
    }

    # create query string
    query_string = urlencode(params)
    logging.info('Query string: {}'.format(query_string))
    # signature
    signature = hmac.new(secret.encode("utf-8"), query_string.encode("utf-8"), hashl

    # url
    url = BASE_URL + '/fapi/v1/order' + "?" + query_string + "&signature=" + signatu

    # post request
    session = requests.Session()
    session.headers.update(
        {"Content-Type": "application/json;charset=utf-8", "X-MBX-APIKEY": key}
    )
    response = session.post(url=url, params={})

    # get order id
    response_map = response.json()
    order_id = response_map.get('orderId')
    print(order_id)
    return order_id

#### 2.3
def get_open_positions(key: str, secret: str, symbol: str):
    # Get the current server time from Binance
    server_time_response = requests.get(BASE_URL + "/api/v3/time")
    server_time = server_time_response.json()["timestamp"]

    # Adjust the timestamp to be 1000ms behind the server time
    timestamp = server_time

    # Position parameters
    params = {
        "symbol": symbol,
        'timestamp': timestamp
    }

    # create query string
    query_string = urlencode(params)
    logging.info('Query string: {}'.format(query_string))

    # signature
    signature = hmac.new(secret.encode("utf-8"), query_string.encode("utf-8"), hashl

    # url
    url = BASE_URL + '/fapi/v2/positionRisk' + "?" + query_string + "&signature=" +

    # Add API key and signature to the request headers
    headers = {
        "X-MBX-APIKEY": key
    }

    # Send the request
    response = requests.get(url, headers=headers)

    # Check if the request was successful
    if response.status_code == 200:
        # Parse the response as JSON
        positions = response.json()
        return positions
    else:
        # If the request was not successful, print the error message
        print("Error:", response.text)
        return None

#### 2.4
```

```
def get_open_orders(key: str, secret: str, symbol: str):

    timestamp = int(time.time() * 1000)
    params = {
        'symbol': symbol,
        'timestamp': timestamp
    }
    query_string = urlencode(params)
    logging.info('Query string: {}'.format(query_string))

    # signature
    signature = hmac.new(secret.encode("utf-8"), query_string.encode("utf-8"), hashl
    # url
    url = BASE_URL + '/fapi/v1/openOrders' + "?" + query_string + "&signature=" + si

    # Define the request headers
    headers = {
        'X-MBX-APIKEY': key
    }

    # Send the request to get all open orders
    response = requests.get(url, headers=headers)

    open_orders = response.json()


    return open_orders

def cancel_all_open_orders(key: str, secret: str, symbol: str):
    timestamp = int(time.time() * 1000)
    params = {
        'symbol': symbol,
        'timestamp': timestamp
    }
    query_string = urlencode(params)
    logging.info('Query string: {}'.format(query_string))

    # signature
    signature = hmac.new(secret.encode("utf-8"), query_string.encode("utf-8"), hashl
    # url
    url = BASE_URL + '/fapi/v1/allOpenOrders' + "?" + query_string + "&signature=" +

    # Define the request headers
    headers = {
        'X-MBX-APIKEY': key
    }
    try:
        # Send the request to get all open orders
        response = requests.delete(url, headers=headers)

        # Check if the request was successful
        if response.status_code == 200:
            print("All open orders cancelled successfully.")

        else:
            print("Failed to cancel all open orders. Status code:", response.status_
    except Exception as e:
        print("An error occurred:", str(e))

    return response.status_code


#### 2.5
def stop_loss(key: str, secret: str, symbol: str, max_trial: int):
    cancel_all_open_orders(api_key, api_secret, 'BTCUSDT')
    positions =\
        get_open_positions(key, secret, symbol)
    order_pos = -float(positions[0]['positionAmt'])
    if order_pos == 0:
        print("No open positions!")
        return

    trial = 0
    while trial < max_trial:
        cancel_all_open_orders(api_key, api_secret, 'BTCUSDT')
        trial += 1
        # Get request to get order book snapshot of BTCUSDT
        response = requests.get('https://fapi.binance.com/fapi/v1/depth', params={'s
        # Get json object from response
        json_object = response.json()
        # call parse() method to convert json message to order book object
        order_book = parse(json_object)

        if order_pos > 0:
            limit_price = order_book.best_ask()

        else:
            limit_price = order_book.best_bid()

        send_limit_order(key,
                         secret,
                         symbol,
                         abs(order_pos),
                         order_pos>0,
                         'GTC',
                         limit_price)
        sleep(20)
        positions =\
            get_open_positions(key, secret, symbol)
        order_pos = -float(positions[0]['positionAmt'])
        if order_pos == 0:
            print(f"Stop Loss at trail No. {trial}")
            return
    print("Limit Order Stop-Loss failed!")
    while order_pos != 0:
        cancel_all_open_orders(api_key, api_secret, 'BTCUSDT')
        send_market_order(key, secret, symbol, abs(order_pos), (order_pos>0))
        sleep(1)
        positions =\
            get_open_positions(key, secret, symbol)
        order_pos = -float(positions[0]['positionAmt'])
    print("Market Order Stop-Loss done!")
    return

#### 2.6
def trade_history(key: str, secret: str, symbol: str):

    timestamp = int(time.time() * 1000)
    params = {
        'symbol': symbol,
        'timestamp': timestamp
    }
    query_string = urlencode(params)
    logging.info('Query string: {}'.format(query_string))

    # signature
    signature = hmac.new(secret.encode("utf-8"), query_string.encode("utf-8"), hashl
    # url
    url = BASE_URL + '/fapi/v1/userTrades' + "?" + query_string + "&signature=" + si

    # Define the request headers
    headers = {
        'X-MBX-APIKEY': key
    }
    response = requests.get(url, headers=headers)
    trade_history = response.json()

    return trade_history



print('All libraries and funcitons loaded')
```

```
All libraries and funcitons loaded
```

In [ ]: