MiniProject

October 11, 2019

1 Import packages

```
In [1]: import pandas as pd
        import struct
        import xlsxwriter
        import operator
        from datetime import timedelta
        from itertools import groupby
```

2 Define variables

```
In [2]: # create a dictionary to store information of added stokes
        stk_list = {}

        #create a dictionary to store all information of traded stokes
        stock_map = {}
```

3 Helper

- Bulid a function to convert nanoseconds to hours

```
In [3]: def nano_to_hour(timestamp):
            n = int.from_bytes(timestamp, byteorder='big')
            s='{0}'.format(timedelta(seconds=n * 1e-9))
            return(int(s.split(':')[0]))
```

4 Add order message and Modify Order Messages

- Use struct.unpack function to unpack binary data.
- The first argument of the function is the format character of the conversion between C and Python values for different message types.

```
In [4]: def add_message(message,msg_type):
            global stk_list
            if msg_type=='A':
```

```
            result=struct.unpack('>HH6sQsI8sI',message)
        if msg_type=='F':
            result=struct.unpack('>HH6sQsI8sI4s',message)

        #add to stoke_list if buy
        if result[4]== 'B':
            order_ref_no = result[3]
            stock_name = result[6].strip()
            stock_price = result[7] / 10000.00
            stk_list[order_ref_no] = (stock_name, stock_price)
        return

    def delete_message(message):
        global stk_list
        result=struct.unpack('>HH6sQ',message)
        order_ref_no = result[3]

        try:
            stk_list.pop(order_ref_no) # remove from stoke list
        except KeyError as e:
            return

    def replace_message(message):
        global stk_list
        result=struct.unpack('>HH6sQQII',message)
        old_order_ref_number = result[3]
        new_order_ref_number = result[4]

        # replace the old stoke with the new one
        try:
            (stock_name, stock_price) = stk_list.pop(old_order_ref_number)
            stk_list[new_order_ref_number] = (stock_name, stock_price)
        except KeyError as e:
            return
        return
```

5 Executed message

```
In [5]: def executed_message(message):
            global stock_map
            global stk_list
            msg_type = 'E'
            result=struct.unpack('>HH6sQIQ',message)

            stock_price = 0
            timestamp = result[2]
            order_ref_no = result[3]
```

```
            share_volume = result[4]
            match_number = result[5]
            hr = nano_to_hour(timestamp)

            try:
                (stock_name, stock_price) = stk_list[order_ref_no]
                if stock_name not in stock_map:
                    stock_map[stock_name] = [(msg_type,hr, order_ref_no, stock_price, share_volu
                else:
                    stock_list = stock_map[stock_name]
                    stock_list.append((msg_type,hr, order_ref_no, stock_price, share_volume))
                    stock_map[stock_name] = stock_list
            except KeyError as e:
                return


        def executed_price_message(message):
            global stock_map
            global stk_list
            msg_type = 'C'
            result=struct.unpack('>HH6sQIQsI',message)

            if result[6] == 'Y':
                timestamp = result[2]
                order_ref_no = result[3]
                share_volume = result[4]
                match_number = result[5]
                stock_price = (result[7]) / 10000.00
                hr = nano_to_hour(timestamp)

                try:
                    (stock_name, stock_price_old) = stk_list[order_ref_no]
                    if stock_name not in stock_map:
                        stock_map[stock_name] = [(msg_type,hr, order_ref_no, stock_price, share_
                    else:
                        stock_list = stock_map[stock_name]
                        stock_list.append((msg_type,hr,order_ref_no, stock_price, share_volume))
                        stock_map[stock_name] = stock_list

                except KeyError as e:
                    return
```

6 Trade Messages

```
In [6]: def trade_message(message):
            global stock_map
            global stk_list
```

```
        msg_type = 'P'
        result= struct.unpack('>HH6sQsI8sIQ',message)

        timestamp=result[2]
        share_volume = result[5]
        stock_name = result[6].strip()
        stock_price=result[7]/10000.00
        match_number = result[8]
        hr = nano_to_hour(timestamp)

        if stock_name not in stock_map:
            stock_map[stock_name] = [(msg_type,hr, match_number, stock_price, share_volume)]
        else:
            stock_list = stock_map[stock_name]
            stock_list.append((msg_type,hr, match_number, stock_price, share_volume))
            stock_map[stock_name] = stock_list

    def broken_trade_message(message):
        global stock_map
        global stk_list
        global exe_orders
        result=struct.unpack('>HH6sQ',message)
        match_number = result[3]
        try:
            (msg_type, order_ref_no, stock_name) = exe_orders.pop(match_number)
            if stock_name in stock_map:
                stock_list = stock_map[stock_name]
                for index, item in enumerate(stock_list):
                    if item[1] == order_ref_no and msg_type == item[0]:
                        del stock_list[index]
                        break
                stock_map[stock_name] = stock_list
        except KeyError as e:
            return

    def cross_trade_message(message):
        global stock_map
        global stk_list
        msg_type = 'Q'
        result= struct.unpack('>HH6sQ8sIQs',message)

        timestamp=result[2]
        share_volume = result[3]
        stock_name = result[4].strip()
        stock_price=result[5]/10000.00
        match_number = result[6]
        hr = nano_to_hour(timestamp)
```

```
            if share_volume == 0:
                return
            elif stock_name not in stock_map:
                stock_map[stock_name] = [(msg_type,hr, match_number, stock_price, share_volume)]
            else:
                stock_list = stock_map[stock_name]
                stock_list.append((msg_type,hr, match_number, stock_price, share_volume))
                stock_map[stock_name] = stock_list
```

7 Parse message

```
In [7]: def parse(message, msg_type):
            if msg_type == 'P':
                trade_message(message)
            elif msg_type == 'C':
                executed_price_message(message)
            elif msg_type == 'E':
                executed_message(message)
            elif msg_type == 'A' or msg_type == 'F':
                add_message(message,msg_type)
            elif msg_type == 'D':
                delete_message(message)
            elif msg_type == 'Q':
                cross_trade_message(message)
            elif msg_type == 'B':
                broken_trade_message(message)
            elif msg_type == 'U':
                replace_message(message)
            else:
                return
```

8 Read and rearrange data

```
In [8]: f = open("D:/Trexquant/data/01302019.NASDAQ_ITCH50",'rb');#read the bindata

        #deal with EOF
        for _ in range(20000000):
            size = int.from_bytes(f.read(2), byteorder='big', signed=False)
            if not size:
                break

            # get the message type for each line
            message_type = f.read(1).decode('ascii')
            record = f.read(size - 1)

            # consider the system event message here
```

```
            if message_type=='S':
                result=struct.unpack('>HH6ss',record)
                # M means end of Market hours
                if result[3]=='M':
                    break

            parse(record, message_type)
```

- Sum up the Quantity and Price × Quantity of different stoke in different time period

```
In [9]: rearrange = {}

        for stoke_name,information in stock_map.items():

            x = information

            # sum up quantity
            sum_q = lambda tu : [(k, sum(u[4] for u in v)) for k, v in groupby(tu, lambda x: x[1
            # sum up price * quantity
            sum_pq = lambda tu : [(k, sum(u[3]*u[4] for u in v)) for k, v in groupby(tu, lambda

            q = sum_q(x)
            pq = sum_pq(x)

            id = operator.itemgetter(0)
            id_inf = {id(rec): rec[1:] for rec in pq}
            new = [info + id_inf[id(info)] for info in q if id(info) in id_inf]

            rearrange[stoke_name] = new
```

9 Output as excel file

```
In [10]: workbook = xlsxwriter.Workbook("D:/Trexquant/data/result.xlsx")
         index = 2

         sheet = workbook.add_worksheet()
         sheet.write('A1', "Stoke name")
         sheet.write('B1', "Hour")
         sheet.write('C1', "Total Volume * Price")
         sheet.write('D1', "Total Volume")
         sheet.write('E1', "VWAP")

         for key, value in rearrange.items():
             for a, item in enumerate(value):
                 sheet.write("A"+str(index), key.decode())
                 sheet.write("B"+str(index), item[0])
                 sheet.write("C"+str(index), item[2])
                 sheet.write("D"+str(index), item[1])
```

```
            sheet.write("E"+str(index), item[2] / (item[1] * 1.00))
            index += 1

    workbook.close()
```

In [11]:
```
# check the result
pd.read_excel('D:/Trexquant/data/result.xlsx').head()
```

Out[11]:

	Stoke name	Hour	Total Volume * Price	Total Volume	VWAP
0	UGAZ	4	59894.36	1547	38.716458
1	UGAZ	5	20887.80	540	38.681111
2	UGAZ	6	6587.60	170	38.750588
3	UGAZ	7	273014.56	7006	38.968678
4	UGAZ	8	10053.85	260	38.668654