Course Project – Part 1(100 pts)

Goal of this project: reading/writing csv files, plotting a chart, creating some calculations (daily return, monthly return) from the data, using data structure

- Unzip the file markedata.zip
- You will find 19 csv files with information about stocks (for the Part I)
- You will find 3 csv files with information about exchange (BARX, EDGX, NYSE) (for the Part 2)

Part I: Reading a CSV file containing Market Information (50 pts)

- Load the content of these files into the following data structure:

market_data['Symbol'] = { ' Open' : [ ] , 'High' : [], 'Low' : [],'Close' : [], 'Volume' : [], 'Adjusted' :[] , 'Date' : []}

Since you have to handle many symbols, *market_data* should be a dictionary of dictionary.

Every values of this dictionary of dictionary will be a list of number or float or date.

Task 1: Create this dictionary

Task 2: Create a new key in this dictionary being a list of values representing the moving average for 10 days.

It means that *market_data* will be:

market_data['Symbol'] = { ' Open' : [ ] , 'High' : [], 'Low' : [],'Close' : [], 'Volume' : [], 'Adjusted' :[] , 'Date' : [], 'MA_10' :[]}

example:

```
>>> market_data = {}

>>> market_data['SYMBOL1'] = { "Open" : [12,13,14,15,16] , "High" : [20,22,23,19,21], "Low" : [10,12,13,9,1] , "Close" : [15,15,15,11,16], "Volume" : [100,200,300,400,500], "Adjusted" :[12,13,14,15,16] , "Date" :['2015-01-01','2015-01-02','2015-01-03','2015-01-04','2015-01-05'], "MA_10" :[]}

>>> market_data['SYMBOL2'] = { "Open" : [12,13,14,15,16] , "High" : [20,22,23,19,21], "Low" : [10,12,13,9,1] , "Close" : [15,15,15,11,16], "Volume" : [100,200,300,400,500], "Adjusted" :[12,13,14,15,16] , "Date" :['2015-01-01','2015-01-02','2015-01-03','2015-01-04','2015-01-05'], "MA_10" :[]}

>>> print market_data
```

{'SYMBOL2': {'High': [20, 22, 23, 19, 21], 'Volume': [100, 200, 300, 400, 500], 'Adjusted': [12, 13, 14, 15, 16], 'Low': [10, 12, 13, 9, 1], 'MA_10': [], 'Date': ['2015-01-01', '2015-01-02', '2015-01-03', '2015-01-04', '2015-01-05'], 'Close': [15, 15, 15, 11, 16], 'Open': [12, 13, 14, 15, 16]}, 'SYMBOL1': {'High': [20, 22, 23, 19, 21], 'Volume': [100, 200, 300, 400, 500], 'Adjusted': [12, 13, 14, 15, 16], 'Low': [10, 12, 13, 9, 1], 'MA_10': [], 'Date': ['2015-01-01', '2015-01-02', '2015-01-03', '2015-01-04', '2015-01-05'], 'Close': [15, 15, 15, 11, 16], 'Open': [12, 13, 14, 15, 16]}}

Since it is a moving average, there are some parts without values; you will just use *None* as a value. All the different lists should have the same length.

Task 3: You will create another key in this dictionary containing the daily return for every day (the first day shouldn't have any return)

Simple return.

For one-period simple **gross return** is

$$1 + R_t = \frac{P_t}{P_{t-1}}$$

or

$$P_t = P_{t-1}(1 + R_t).$$

The corresponding one-period simple **net return** is

$$R_t = \frac{P_t}{P_{t-1}} - 1 = \frac{P_t - P_{t-1}}{P_{t-1}}.$$

You will calculate the return using the adjusted price (the adjusted price taking into account split and dividends)

Task 4: You will create:

- A function 'maximum_return', which will return the best return across all the symbols. The return should contain the return and the date associated
- A function 'minimum_return', which will return the worst return across all the symbols. The return should contain the return and the date associated
- A function 'best_return_for_one_month', which will return the best return across all the symbols for a 1-month period. The return should contain the return and the month/year

Task 5: Plot a chart for MSFT, NVDA, GOOGL representing the daily market price.

You will use the library the library matplot.

```
import matplotlib
import matplotlib.pyplot as plt
```

```
plt.plot([1,2,3,4])
plt.ylabel('some numbers')
plt.show()
```

Part II: Creating an order book (50pts)

You are creating an order book builder for your trading system. The book builder will have as an input a list of files with the same format. Each exchange will have a given file. During this exercise, you will take 3 venues (3 files) as input: ARCA, EDGX, NYSE.

You will need to read the three files at the same time (to make it deterministic, you will always read ARCA first then EDGX second).

An order book is defined as the following on wiki:
https://en.wikipedia.org/wiki/Order_book_(trading)

A book builder is a component of a trading system sorting the orders from different exchanges by price and by side. It is a critical component: Primary source of market information for trading models.

Example of an order book:

(You will not need to use the ID column)

BID					OFFER				
ID	Venue	Time	Volume	Price	Price	Volume	Time	Venue	ID
2	ARCA	7/4/16 11:25	1	110	90	4	7/4/16 11:25	EDGX	1
					100	2	7/4/16 11:25	NYSE	3

For the id, since you will have only 3 exchanges for two sides (BID,OFFER), maybe you can consider having this ID encoding:
Orderid(NYSE-BID): 11
Orderid(NYSE-OFFER): 21
Orderid(EDGX-BID): 12
Orderid(EDGX-OFFER): 22
Orderid(ARCA-BID): 13
Orderid(ARCA-OFFER): 23

Task 1: Create the class *orderbook*.

You can use the following example:

```
class orderbook:
    bid={}
    offer={}
    def __init__(self):
        self.bid['EXCHANGE1']={'price':0, 'quantity':0 }
        self.bid['EXCHANGE2']={'price':0, 'quantity':0 }
        self.bid['EXCHANGE3']={'price':0, 'quantity':0 }
```

```
        self.offer['EXCHANGE1']={'price':0, 'quantity':0 }
        self.offer['EXCHANGE2']={'price':0, 'quantity':0 }
        self.offer['EXCHANGE3']={'price':0, 'quantity':0 }

    def process_tick(self,order):
        if order['side']=='BID':
            self.bid[order['exchange']].update({'price' : order['price'], 'quantity' :
order['quantity']})
        else:
            self.offer[order['exchange']].update({'price' : order['price'], 'quantity' :
order['quantity']})
        self.top_of_book()

    def top_of_book(self):
        # You need to send the book sorted
        # I am returning random
        # you need to sort
        print '-----'
        print "BID :%s %s %s" % ('EXCHANGE1', self.bid['EXCHANGE1']['price'],
self.bid['EXCHANGE1']['quantity'])
        print "OFFER :%s %s %s" % ('EXCHANGE2', self.offer['EXCHANGE2']['price'],
self.offer['EXCHANGE2']['quantity'])
        print '-----'



ob1 = orderbook()
order={'price':1.23, 'quantity':100, 'exchange' : 'EXCHANGE1', 'side' : 'BID'}
ob1.process_tick(order)

order={'price':1.25, 'quantity':100, 'exchange' : 'EXCHANGE1', 'side' : 'SELL'}
ob1.process_tick(order)

order={'price':1.22, 'quantity':50, 'exchange' : 'EXCHANGE2', 'side' : 'SELL'}
ob1.process_tick(order)

order={'price':1.22, 'quantity':150, 'exchange' : 'EXCHANGE2', 'side' : 'SELL'}
ob1.process_tick(order)
```

This is not the class you should implement, some member variables are missing.
This is an example to show you how to process the file line by line.
Each time you get a line from one of the files, you will convert this line into a structure that your function *process_tick* will be able to process. Every time, you will process a line, you will return the Top of the Book. In other words, you will return the best bid/offer (price/quantity/exchange_name). This information is needed to trade.

In this Task1, you will create the class orderbook with the function process_tick and top_of_book. process_tick calling top_of_book once the new string has been processed.

To test your code, we will generate the output of your code running to check if your book is the same as the one we will use as reference.

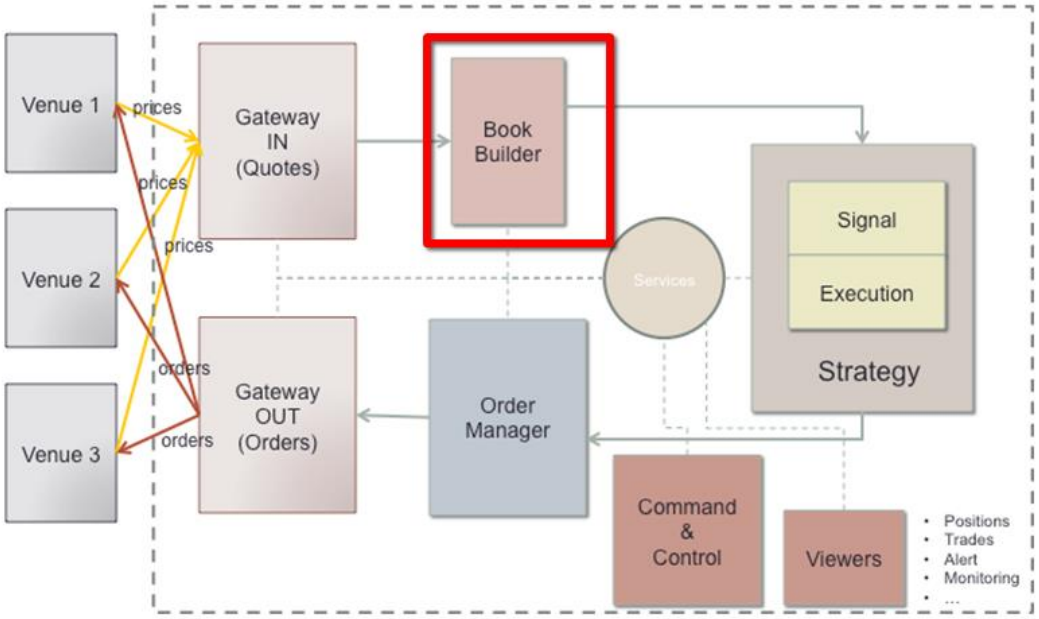
Be aware: some dates are missing in the three files. You should be sure to update with the right timestamp.

For instance: suppose you have:
NYSE-Tick1 at 1.3s
EDGEX-Tick2 at 1.2s
ARCA-Tick3 at 1.1s

You will process ARCA-Tick3 first, then you will need to process EDGEX-Tick2 at 1.2s if the next tick from ARCA is not earlier than 1.2s.
An advice for this part would be to gather these 3 files into 1 and sort by the date (using python). You still need to read the three files with python (please don't use excel to gather the 3 files into 1).

Task 2: You will create the function.
getBestBid retuning the best bid (price, volume, exchange)
getBestOffer retuning the best offer (price, volume, exchange)

Book Builder



In this assignment, you will read the files and get the orders coming from the different exchanges.
You will create a class capable to handle the market data coming one by one and build a book for each side. In this part I, you will just need to handle one symbol for 3 different exchanges. But in the following part, you will have a book for different symbols, therefore it is important to have your book handling many symbols.

Execution example (Cf. Lecture October 18th)

Cf. Piazza for examples

Task 3: You will create the function:
getBidVolumeBetween(price1, price2) returning the total volume between price1 and price2 for bids.
getOfferVolumeBetween(price1, price2) returning the total volume between price1 and price2 for offers