

FE 520C Assignment Three

March 9, 2023

Submission Requirement:

1. For all the problems in this assignment you need to design and use Python 3, output and present the results in a nice format.
2. In this assignment, you are going to try writing **class** in python. Therefore, you are supposed to submit two kinds of files for this homework: one Jupyter notebook (**.ipynb**) file and three Python script (**.py**) files (two .py files for question 1, one .py file for question 2).
3. You are strongly encouraged to write comment for your code, because it is a convention to have your code documented all the time. In your python files, you need contain both function and test part of function.
4. Do NOT copy and paste from others, all homework will be firstly checked by plagiarism detection tool.

Question 1 2D Random Walks (50 pts)

Implement a 2D random walk class. 2D random walk is a process that a point (with coordinates x and y) walks randomly on a 2D axis. For each time step $t = 1, 2, 3, \dots$, you flip a **fair coin** twice and obtain the result C_1 and C_2 , where C_1 and C_2 could be head (H) or tail (T). Then, you move the points from (x_{t-1}, y_{t-1}) to (x_t, y_t) using the following rule:

$$x_t = \begin{cases} x_{t-1} + 1 & C_1 = H \\ x_{t-1} - 1 & C_2 = T \end{cases} \quad \text{and} \quad y_t = \begin{cases} y_{t-1} + 1 & C_1 = H \\ y_{t-1} - 1 & C_2 = T \end{cases}$$

whose initial state by default is $(x_0, y_0) = (0, 0)$. Each sample path of 2D random walk is represented as a list of 2-element tuples. E.g. $[(0, 0), (-1, 1), (0, 2), (1, 3), \dots]$.

Create a Python file called **Random.Walk.py**. You need to implement this 2D random walk **class** in this file. Name your the class as **2D_random_walk**. The initializer should be one parameter **init_state**. The class should have one method **generate(num_path, num_step)** to output a list of paths, where **num_path**, **num_step** is respectively the number of paths and the length of each path you want to generate. The generated path should always start from the initial state.

As an example, your class should work with the following calls:

```

if __name__ == "__main__":
    rw = Random_Walk_2D(init_state = [0,0])
    rw.generate(num_path = 2, num_step = 3)
    # returns:
    # [[(0,0), (-1,1), (0,2)],
    #  [(0,0), (1,1), (2,0)]]
    # random results,
    # your results could be different from this

```

1. (40 pts) Implement the aforementioned class.
2. (10 pts) Create a script `main.py`, import and test your class with the following input:

```

if __name__ == "__main__":
    from Random_Walk import Random_Walk_2D
    rw = Random_Walk_2D(init_state = [5,5])
    rw.generate(num_path = 10, num_step = 100)
    rw.generate(num_path = 5, num_step = 50)

```

Question 2 Stock Class (50 pts)

2.1 (10 pts) Build A Stock Class

Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. In this practice, we are going to build a class for stock and define methods to perform analysis. Specifically, the “Stock” has the following attributes: “ticker”, “open price”, “close price”, “volume”, “shares”. Stock objects also have methods: calculate daily `return()`, calculate `cost()`. Please implement these two methods. (**Note: assume we buy the stock the moment the market opens and hold the stock.**)

After creating the Stock class, please create a list of instances using the information from the following portfolio list. (**Note: you are required to use just one line of code to implement this step.**)

```

portfolio = [
    {'ticker': 'IBM', 'shares': 100, 'open price': 91.1, 'close price' : 103, 'volume': 100},
    {'ticker': 'AAPL', 'shares': 50, 'open price': 543.22, 'close price' : 653.1, 'volume': 200},
    {'ticker': 'FB', 'shares': 200, 'open price' : 21.09, 'close price' : 25.2, 'volume': 150},
    {'ticker': 'HPQ', 'shares': 35, 'open price' : 31.75, 'close price' : 43.89, 'volume' : 120},
    {'ticker': 'YHOO', 'shares': 45, 'open price' : 16.35, 'close price' : 17.23, 'volume': 87},
    {'ticker': 'ACME', 'shares': 75, 'open price' : 115.65, 'close price' : 120.3, 'volume': 86}]

```

2.2 (20 pts) Analyze The Portfolio

In the previous step, we create a list that contains several stocks. We can consider it as a “portfolio”. In this practice, we are going to calculate the basic statistics of the portfolio and do some basic analysis.

1. Use one line of code to calculate the total shares in the portfolio
2. Use one line of code to calculate the total cost in the portfolio
3. Use one line of code to build a list based on the portfolio (i.e., the list of instances we obtain in the above). Each element of the list is a dictionary. Each dictionary have two key-values pairs for each stock in the portfolio. The keys are ticker and open price. The value are the corresponding ticker name and open price of each stock. Let's call this list ticker price list.
4. Use one line of code to sort the ticker price list based on the price.
5. Use one line of code to find the stock with largest open price in the ticker price list.
6. Use one line of code to find the highest 3 stocks in terms of close price in the ticker price list.
(hint: we are going to use the `heapq` package and `heapq.nlargest()` method. The parameter we can input for method: `heapq.nlargest(number of largest number, iterable, criteria)`)

2.3 (20 pts) Class Inheritance

1. Define an inheritance `StockInfo` based on the `Stock` class. The new attributions for this class are default times and years. Please use `super()` method for initialization.
2. Please define a method called `StockInfo.update_years()`. When you call this method, the "years" attribute will add one to itself.
3. Please define a class variable called penalty threshold and let it equal to 5. (Class variable is a variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods.) Then override the method `calculate_return()`. If the default time is greater than the penalty threshold, the return will be deduct 10%.
4. Define another class called `VolumeLevel` with attribute `volume` and method `determine volume level()`. If `volume` is greater than 100, then it will be classified as high. Otherwise it will be classified as low. Make instance of `VolumeLevel` as attribute of class `StockInfo`. Then call the method `determine volume level()`.
5. For now when you try to print the instance of `StockInfo`, you will get the information of the class. Please use magic method `repr()` to change the output information. When you print the instance, you will get the information of the stock.
The format is: `ticker-open price-shares`.