Task 1 : Interface with a stock price data feed

Interface with a stock price data feed and set up your system for analysis of the data.

Step 1: Video

Task 1 Final

Step 2 : Background Information of the task :

You've been asked to assist with some development to add a chart to a trader's dashboard allowing them to better identify under/over-valued stocks.

The trader would like to be able to monitor two historically correlated stocks and be able to visualize when the correlation between the two weakens (i.e. one stock moves proportionally more than the historical correlation would imply). This could indicate a potential trade strategy to simultaneously buy the relatively underperforming stock and sell the relatively outperforming stock. Assuming the two prices subsequently converge, the trade should be profitable.

Most data visualization for our traders is built on JPMorgan Chase's Perspective data visualization software, which is now open source. If you want to explore that, a link is provided in the resources section.

Before implementing this request using perspective, first you'll need to interface with the relevant financial data feed and make the necessary adjustments to facilitate the monitoring of potential trade opportunities.

- Understanding the finance and trading part is not required.
- Being familiar with python scripting language and command line basics is not required as you will be guided in this exercise
- Note, you DO NOT have to install Perspective as an individual software onto your machine. All you need to complete this task is to follow the instructions in step 3)

Step 3: Tasks to be followed

For the first module of this project will need you to accomplish the following:

1. Set up your system by downloading the necessary repository, files, tools and dependencies
2. Fix the broken client datafeed script in the repository by making the required adjustments to it.
3. Generate a patch file of the changes you made
4. Bonus task: Add unit tests in the test script in the repository.

We've broken down the steps for you in stages below so you can accomplish this task in an organized manner.

Set Up

Before you can tackle any software or development task you need to set up your development environment. Your development environment refers to your system having all the required software installed to modify the code, as well as getting the code of the project itself onto your computer.

To do this we've created a simple PDF guide below on how to get your environment set up:

- Setting up your dev environment -- setup_devenv_m1_v6.pdf
- REPL environment for python 3 or higher

Step 4 : Making Changes

Making Changes

When you're in a work environment, you'll usually receive tasks in the form of engineering tickets. Here is an example of what this task looks like in the form of an engineering ticket

Purpose
We want to process the data feed of stock A and stock B's price to enable us to analyse when trading for the stock should occur.

Acceptance Criteria

- `getDataPoint` function should return correct tuple of stock name, bid_price, ask_price and price. Note: price of a stock = average of bid and ask
- `getRatio` function should return the ratio of the two stock prices
- `main` function should output correct stock info, prices and ratio
- Upload a git patch file as the submission to this task
- Bonus: All unit tests inside client_test.py, added/existing have to pass

Task Step by step guidelines :

To properly make the changes necessary to complete the objectives of this task, follow the guide below.

- Step by Step REPL guidelines is given here -> (repl_mod1_v4.pdf)
- For using Windows / Mac instead of REPL environment, follow this guideline -> making_changes_m1_v4a.pdf
- For bonus task, read this -> client_test_m1_v1a.pdf

Before Start Coding (Initialization Git Repository)

```
import os
os.system('bash')
cd jpm_module_1
git init
git add -A
git config user.email "<your_email_address>"
git config user.name "<your_name>"
git commit -m 'INIT'
exit
```

Objective

Initial (wrong) Output

```
Quoted ABC at (bid:127.59, ask:128.84, price:127.59)
Quoted DEF at (bid:124.66, ask:124.8, price:124.66)
Ratio 1
Quoted ABC at (bid:127.59, ask:128.84, price:127.59)
Quoted DEF at (bid:124.66, ask:124.8, price:124.66)
Ratio 1
Quoted ABC at (bid:127.59, ask:128.84, price:127.59)
Quoted DEF at (bid:124.66, ask:124.8, price:124.66)
Ratio 1
Quoted ABC at (bid:127.59, ask:125.05, price:127.59)
Quoted DEF at (bid:124.66, ask:125.15, price:124.66)
Ratio 1
>
```

If we closely inspect the output, there are two incorrect thing :-
(1) Ratio is always 1
(2) The price of each stock is always the same as its `bid_price`.

Final target Output

```
Quoted ABC at (bid:127.59, ask:128.84, price:128.215)
Quoted DEF at (bid:124.66, ask:124.8, price:124.72999999999999)
Ratio 1.0279403511585026
Quoted ABC at (bid:127.59, ask:128.84, price:128.215)
Quoted DEF at (bid:124.66, ask:124.8, price:124.72999999999999)
Ratio 1.0279403511585026
Quoted ABC at (bid:127.59, ask:128.84, price:128.215)
Quoted DEF at (bid:124.66, ask:124.8, price:124.72999999999999)
Ratio 1.0279403511585026
Quoted ABC at (bid:127.59, ask:125.05, price:126.32)
Quoted DEF at (bid:124.66, ask:125.15, price:124.905)
Ratio 1.0113286097434049
UNIT TEST RESULTS BELOW...
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

Making changes in `client.py` (client3.py for python3) `getDataPoint()`

`getDataPoint()` In this method, you'll have to make the modifications to compute for the right stock price. This means you have to change how `price` is computed for. The formula is `(bid_price+ask_price) / 2`.
YOU DO NOT NEED TO CHANGE the return value as that is representational of the entire data point. You should end up with something like:

```
def getDataPoint(quote):
    """ Produce all of the needed values to generate a datapoint """
    """ ------------- Update this function ------------- """
    stock = quote['stock']
    bid_price = float(quote['top_bid']['price'])
    ask_price = float(quote['top_ask']['price'])
    price = (bid_price + ask_price)/2
    return stock, bid_price, ask_price, price
```

Making changes in `client.py` (client3.py for python3) `getRatio()`

`getRatio()` In the original case, this method just returns 1 all the time. To correct this, you must change the return value to the ratio of stock price_a to stock price_b

```
def getRatio(price_a, price_b):
    """ Get ratio of price_a and price_b """
    """ ------------- Update this function ------------- """
    """ Also create some unit tests for this function in client_test.py """
    if(price_b==0):
        return
    return price_a/price_b
```

Making changes in `client.py` (client3.py for python3) : `main()`

`main()` method: Now that you've fixed the two other methods, it's just a matter of printing the correct values. For every iteration in the main method, you need to store the datapoints you get from getDataPoint method so that you can properly call getRatio and print the right ratio out:

```
# Main
if __name__ == "__main__":

    # Query the price once every N seconds.
    for _ in range(N):
        quotes =
json.loads(urllib.request.urlopen(QUERY.format(random.random())).read())

        """ ----------- Update to get the ratio --------------- """
        prices = {}
        for quote in quotes:
            stock, bid_price, ask_price, price = getDataPoint(quote)
            prices[stock] = price
            print ("Quoted %s at (bid:%s, ask:%s, price:%s)" % (stock,
bid_price, ask_price, price))

        print ("Ratio %s" % getRatio(prices['ABC'], prices['DEF']))
```

Creating Patch file when done. (If want to do bonus task, see [below](#))

```
# Input this to bash shell one by one and press enter
import os
os.system('bash')
cd jpm_module_1
git add -A
git commit -m 'Create Patch File'
git format-patch -1 HEAD
mv <name of patch created> ../.
exit
```

Bonus Task Details: `client_test.py`

In the case of this task, the methods we want to test are getDataPoint and getRatio. We want to cover these with tests so that we ensure these methods are returning the correct output as individual methods thereby ensuring us that when we use them together in the main client application, things will work fine.

- In unit testing, we always follow the pattern, Build-Act-Assert:
  - **Build:** We first build a simulated test scenario e.g. instantiating the dummy data we will pass in the methods we'll test, importing the class whose methods we want to test, etc.
  - **Act**: We then make some operations and call the method we want to test for
  - **Assert**: We check if the output of the method we're testing matches the expectation we have (e.g. dummy / static data of the outcome)
- Unit tests mostly use static / dummy data to represent the test cases we pass into methods and the expected outcome of these methods when such test cases are given. This is fine because we want our methods and tests to be deterministic and not randomly failing / behaving in a different way we don't expect.
- For this part of the task, you atleast have to add assertions to the test methods that already exist in the `client_test.py` file.
- An assertion would look something like the following bullet points below:

```
self.assertEqual(getDataPoint(quote),dataPoint)
self.assertEqual(1,1)
```

- For more examples on assertions in python unit testing check this resource or other online resource about Python unit testing.

Testing Assertion Code :

```
for quote in quotes:
    self.assertEqual(getDataPoint(quote), (quote['stock'], quote['top_bid']
['price'], quote['top_ask']['price'], (quote['top_bid']['price'] +
quote['top_ask']['price'])/2))
```

Make a patch file (create_patch_file_v3a.pdf) and submit here

---

Required Final Codes

final client3.py

```
#############################################################################
#
#  Permission is hereby granted, free of charge, to any person obtaining a
#  copy of this software and associated documentation files (the "Software"),
#  to deal in the Software without restriction, including without limitation
#  the rights to use, copy, modify, merge, publish, distribute, sublicense,
#  and/or sell copies of the Software, and to permit persons to whom the
#  Software is furnished to do so, subject to the following conditions:
#
#  The above copyright notice and this permission notice shall be included in
#  all copies or substantial portions of the Software.
#
#  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
#  OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
#  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
#  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
#  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
#  FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
#  DEALINGS IN THE SOFTWARE.

import urllib.request
import time
import json
import random

# Server API URLs
QUERY = "http://localhost:8080/query?id={}"

# 500 server request
N = 500

def getDataPoint(quote):
    """ Produce all of the needed values to generate a datapoint """
    """ ------------- Update this function ------------- """
    stock = quote['stock']
    bid_price = float(quote['top_bid']['price'])
    ask_price = float(quote['top_ask']['price'])
    price = (bid_price + ask_price)/2
    return stock, bid_price, ask_price, price

def getRatio(price_a, price_b):
    """ Get ratio of price_a and price_b """
    """ ------------- Update this function ------------- """
    """ Also create some unit tests for this function in client_test.py """
    if(price_b==0):
        return
    return price_a/price_b

# Main
if __name__ == "__main__":
```

```
    # Query the price once every N seconds.
    for _ in range(N):
        quotes =
json.loads(urllib.request.urlopen(QUERY.format(random.random())).read())

        """ ----------- Update to get the ratio --------------- """
        prices = {}
        for quote in quotes:
            stock, bid_price, ask_price, price = getDataPoint(quote)
            prices[stock] = price
            print ("Quoted %s at (bid:%s, ask:%s, price:%s)" % (stock,
bid_price, ask_price, price))

        print ("Ratio %s" % getRatio(prices['ABC'], prices['DEF']))
```

final main.py

```
# BEFORE YOU DO ANYTHING, PLEASE DO FF
# CLICK THE 'FORK' BUTTON ABOVE

# READ`Instructions` file. You can access it by clicking on the 'Files' icon on
the left side/column of the screen (i.e. the top most icon, above Packages and
Settings) and then clicking on the 'Instructions' file to show the contents

# This is the main.py script. It's the only script that you can run on this REPL
platform via the run button above. This script takes care of displaying the
output of stock price feed script you need to change in the `jpm_module_1`
folder.

# YOU SHOULD NOT CHANGE ANYTHING HERE AS IT ONLY RUNS THE STOCK PRICE FEED
SCRIPT AND SHOWS THE OUTPUT

# IF YOU WISH TO DO THE BONUS TASK DESCRIBED IN THE INSRUCTIONS FILE, UNCOMMENT
THE CODE BELOW

import os
import subprocess
import time
import signal

os.chdir(os.getcwd()+'/jpm_module_1')

process = subprocess.Popen(['python', 'server3.py'], cwd=os.getcwd(),
preexec_fn=os.setsid)

time.sleep(.300)

process2 = subprocess.Popen(['python', 'client3.py'], cwd=os.getcwd(),
preexec_fn=os.setsid)
process2.wait()
os.killpg(os.getpgid(process.pid), signal.SIGTERM)

# FOR BONUS TASK
# IF YOU WANT TO DO IT THEN UNCOMMENT THE CODE BELOW
# Comments are anything that's preceded with '#'
```

```
# TO UNCOMMENT JUST REMOVE THE '#'

print("UNIT TEST RESULTS BELOW...")
process2 = subprocess.Popen(['python', 'client_test.py'], cwd=os.getcwd(),
preexec_fn=os.setsid)
process2.wait()
```

final client_test.py

```
import unittest
from client3 import getDataPoint, getRatio

class ClientTest(unittest.TestCase):
  def test_getDataPoint_calculatePrice(self):
    quotes = [
      {'top_ask': {'price': 121.2, 'size': 36}, 'timestamp': '2019-02-11
22:06:30.572453', 'top_bid': {'price': 120.48, 'size': 109}, 'id':
'0.109974697771', 'stock': 'ABC'},
      {'top_ask': {'price': 121.68, 'size': 4}, 'timestamp': '2019-02-11
22:06:30.572453', 'top_bid': {'price': 117.87, 'size': 81}, 'id':
'0.109974697771', 'stock': 'DEF'}
    ]
    """ ------------ Add the assertion below ------------ """
    for quote in quotes:
      self.assertEqual(getDataPoint(quote), (quote['stock'], quote['top_bid']
['price'], quote['top_ask']['price'], (quote['top_bid']['price'] +
quote['top_ask']['price'])/2))

  def test_getDataPoint_calculatePriceBidGreaterThanAsk(self):
    quotes = [
      {'top_ask': {'price': 119.2, 'size': 36}, 'timestamp': '2019-02-11
22:06:30.572453', 'top_bid': {'price': 120.48, 'size': 109}, 'id':
'0.109974697771', 'stock': 'ABC'},
      {'top_ask': {'price': 121.68, 'size': 4}, 'timestamp': '2019-02-11
22:06:30.572453', 'top_bid': {'price': 117.87, 'size': 81}, 'id':
'0.109974697771', 'stock': 'DEF'}
    ]
    """ ------------ Add the assertion below ------------ """
    for quote in quotes:
      self.assertEqual(getDataPoint(quote), (quote['stock'], quote['top_bid']
['price'], quote['top_ask']['price'], (quote['top_bid']['price'] +
quote['top_ask']['price'])/2))

  """ ------------ Add more unit tests ------------ """



if __name__ == '__main__':
    unittest.main()
```