DS-GA-1007 Programming for Data Science

Assignment 8

Submission Instructions

You are free to use whichever development environment you wish to create and submit the assignment answers.

1. Create a directory using your *Net ID* as the directory name.
2. Place your Python code in this directory.
3. There should be at least one file called `assignment8.py` at the top level of this directory. This is the main program that will generate your answers.
4. Fork the `assignment8` repository from the ds-gs-1007 user on GitHub.
5. Clone this repository onto your local system.
6. Place your new directory (the Net ID) into the working directory of this repository either using PyDev or manually.
7. Add your directory to the staging area, commit, and push to the remote repository.
8. Submit a pull request to the repository owner (ds-ga-1007).

Questions

1. Suppose we have an investment instrument with the following properties:

    a. You can purchase it in $1, $10, $100, and $1000 denominations.

    b. Holding time is one day.

    c. 51% of the time the return is exactly 1.0 (the value doubles).

    d. 49% of the time the return is exactly -1.0 (all value is lost).

    This "investment instrument" is like an even money bet on a biased coin (that comes up "heads" 51% of the time). The odds for this game are very similar to the odds held by the casino for even money bets at roulette.

    Suppose further that we have $1000 to invest on the first day.

This assignment will run a simulation to determine how to make that investment on the first day. i.e. Should we make a single $1000 investment, or 1000 $1 investments (or something in between)?

2. Create a Python program that will do the following:

   a. Accept the following inputs from the user:

`positions`	a list of the number of shares to buy in parallel: e.g. `[1, 10, 100, 1000]`
`num_trials`	how many times to randomly repeat the test

   b. For each position, set a value to represents the size of each investment

   ```
   position_value = 1000 / position
   ```

   c. Use NumPy's random number generating capability to simulate the outcome of one day of investment:

   - Call the result `cumu_ret[trial]`
   - Example for the case where `position_value = 1000`, the outcome should be 0 (49% chance) or $2000 (51% chance)

   d. Repeat `num_trials` times (e.g., simulate 10,000 different single days of trading).

   e. Save the result of each day as:

   ```
   daily_ret[trial] = (cumu_ret[trial]/1000) - 1
   ```

3. Run your program with positions set to `[1, 10, 100, 1000]` and `num_trials` set to `10000.`

   For the run, compute results as follows:

   a. For each position, plot of the result of the trials in a histogram with X axis from -1.0 to +1.0, and Y axis as the number of trials with that result. [Hint: use the matplotlib function `plt.hist(daily_ret,100,range=[-1,1])`]

   b. For each position, the mean or expected value of the daily return.

   c. For each position, the standard deviation of the daily return.

4. The program should generate five files:

`results.txt`	The numerical results described above
`histogram_0001_pos.pdf`	The histogram of the result for 1 position of $1000
`histogram_0010_pos.pdf`	The histogram of the result for 10 positions of $100
`histogram_0100_pos.pdf`	The histogram of the result for 100 positions of $10
`histogram_1000_pos.pdf`	The histogram of the result for 1000 positions of $1

Grading

This assignment will be graded according to the criteria listed in the following 5 sections.

Correctness

- The program produces the correct output when run using the command

```
python assignment8.py
```

Exception/Error Handling

- All possible exceptions are handled correctly
- The code catches specific exceptions (e.g. `KeyboardInterrupt`) rather than using a catchall statement
- Invalid user input is handled correctly (when input is required by the assignment)
- User defined exception(s) are employed for indicating error conditions rather than raising generic exceptions

Comments

- The main program contains a comment that lists the authors, and describes the overall program behavior
- Comments are used to explain intent and/or warn of consequences where appropriate
- Doc strings are used to describe each function
- Comments are used to document public methods in the class
- There is no commented-out code

Structure

- At least one class is used
- The class is in separate module from main program
- Modules are used to structure the program
- The program is correctly structured as a Python package

- The code is easily understandable (i.e. divided into logical sections, well structured, etc.)
- The code uses meaningful names for variables, functions, and methods, and avoids “Hungarian” notation
- Function/method bodies are kept small

Testing

- Unit tests are provided with the solution code
- The unit tests pass correctly