UDACITY

‹ Return to Classroom

Smart Beta Portfolio and Portfolio Optimization

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Congratulations on meeting all the specifications of the project! It's been pleasure reviewing your project! In most of the functions, you have mostly tried to avoid using any loops to perform any operations on to the Pandas Series/DataFrame which is quite inefficient. Pandas is an excellent data analysis and manipulation library which scales quite well if implemented correctly. You have implemented Pandas library correctly.

Part 1: Smart Beta Portfolio

The function `generate_dollar_volume_weights` computes dollar volume weights.

Nice start! 👍

The function `calculate_dividend_weights` computes dividend weights.

Nice work in getting the weights of the total dividend yield over time. The sum of the prices of the stocks over each time-step has been divided by the sum of the portfolio prices over the time period.

The function `generate_returns` computes returns.

You can put the one liner of code by using `pct_change()` method of Pandas

The function `generate_weighted_returns` computes weighted returns.

The function `calculate_cumulative_returns` computes cumulative returns.

Also, check out this Knowledge Forum Post for this function;
https://knowledge.udacity.com/questions/16359

The function `tracking_error` computes tracking error.

Nice work in using the `std()` method on the dataframe. Here Pandas DataFrame/Series `std()` method already uses `ddof=1` by default https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.std.html

You can also use `np.std` with `ddof=1` through which you calculate the sample standard deviation where divisor is `N - 1` as `ddof=1` https://docs.scipy.org/doc/numpy/reference/generated/numpy.std.html

Part 2: Portfolio Optimization

The function `get_covariance_returns` computes covariance of the returns.

Nice job of calculating the portfolio variance (i.e. covariance returns) which needs to be minimize in the objective function of the portfolio optimization problem

The function `get_optimal_weights` computes optimal weights.

Nice job implementing the objective function (`xTPx+λ ‖ x-index ‖ 2`) and minimizing it given the constraints (`[x >= 0, sum(x) == 1]`) to get the optimal ETF weights. Check out the docs https://www.cvxpy.org/

`cvxpy` uses Disciplined Convex Programming (DCP) to ensure that the specified optimization problems are convex. If you get an error about DCP, it likely means that there is something wrong with the way you specified your problem. You can check that a problem, constraint, or objective satisfies the DCP rules by calling `object.is_dcp()` in order to narrow down any of possible the root of your issue. For example;

```
print(cvx.Problem(objective, constraints).is_dcp())
```

The function `rebalance_portfolio` computes weights for each rebalancing of the portfolio.

With the `rebalance_portfolio`, the ETF portfolio remains optimized throughout the entire period

The function `get_portfolio_turnover` computes cost of all the rebalancing.

`portfolio_turnover` calculates the proportional to the magnitude of change in the portfolio weights.

DOWNLOAD PROJECT

RETURN TO PATH