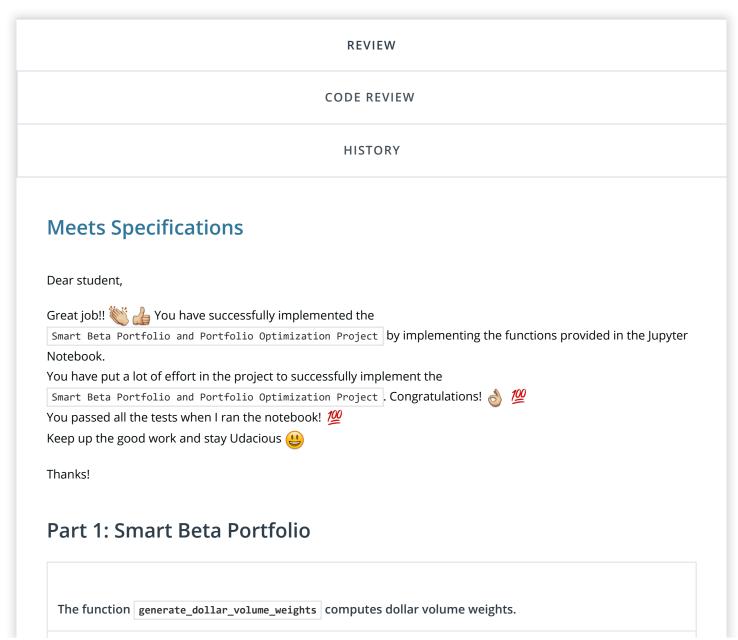


## < Return to Classroom

## Smart Beta Portfolio and Portfolio Optimization



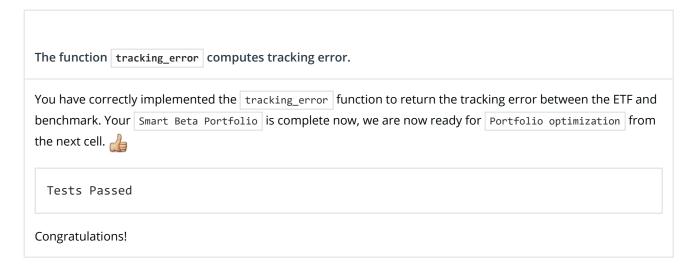
You have correctly calculated the dollar volume to generate dollar volume weights in the generate\_dollar\_volume\_weights function. Normalization of values is done correctly for the sum of the total dollar-volume of the stocks traded every day 👍. Tests Passed Congratulations! An alternative way to implement | generate\_dollar\_volume\_weights | is as following: return (close\*volume).apply(lambda x: x/x.sum(),axis=1)

The function calculate\_dividend\_weights computes dividend weights. You have correctly implemented the calculate\_dividend\_weights function by calculating the total dividend yield over time. 👌 Tests Passed Congratulations! An alternative way to implement calculate\_dividend\_weights is as following: return dividends.cumsum().apply(lambda x:x/x.sum(),axis=1)

The function generate\_returns computes returns. You have correctly implemented the generate\_returns function to generate returns data for all the stocks and dates from price data. You would have noticed that we're implementing returns and not log returns because we're not dealing with volatility, hence we don't have to use log returns. 👍 Tests Passed Congratulations! An alternative way to compute returns is as follows return prices.pct\_change()

The function generate_weighted_returns computes weighted returns.
You have implemented the <code>generate_weighted_returns</code> function correctly to generate weighted returns using the simple multiplication of returns and weights.
Tests Passed
Congratulations!

The function calculate\_cumulative\_returns computes cumulative returns. You have implemented the calculate\_cumulative\_returns correctly to calculate the cumulative returns over time given the returns. We can use the cumulative returns to compare the performance between the ETF and index. 🐃



## Part 2: Portfolio Optimization

The function get\_covariance\_returns computes covariance of the returns. You have correctly implemented the <code>get\_covariance\_returns</code> to calculate the covariance of the <code>returns</code> which can be used to calculate the portfolio variance. Tests Passed Congratulations!

The function get_optimal_weights computes optimal weights.
You have correctly optimized the weights using cvxpy to implement the get_optimal_weights function.  Variable, Objective and Constraints are set properly to solve the optimization problem with the help of
cvx.Problem(objective, constraints).solve() . You have also returned x.value correctly in the end.
Tests Passed
Congratulations!
The function rebalance_portfolio computes weights for each rebalancing of the portfolio.
You have correctly rebalanced the portfolio over the same period instead of using the same weights for the entire history. You have rebalanced the portfolio every n number of days, using shift_size. When rebalancing you also looked back at a certain number of days of data in the past, denoted as chunk_size. You have also computed the optimal weights using get_optimal_weights and get_covariance_returns functions.
Tests Passed
Congratulations!
The function get_portfolio_turnover computes cost of all the rebalancing.
You have now successfully implemented Smart Beta Portfolio which is evident by the impressive portfolio turnover value of 16.594080020340048 you have achieved from your portfolio. However, we need to be cognizant that this is a simulated market cap weighted index with large dollar volume stocks only which is not comparable one on one with the real world portfolios.
Tests Passed
Congratulations!

## **■** DOWNLOAD PROJECT

6/15/2021

RETURN TO PATH

Rate this review START