In The Name of God

# Final Project:
# Optimization of Portfolio

Group Members:

MohammadAli Khosroabadi

Pedram Khojaste Rad

## Cell 1 (imports):

Importing Necessary Packages and Libraries for Code

## Cell 2 (fetch_data):

**Downloading Data & Labeling it with file_name:**

- **`file_name`** is created for later identification, to prevent repeated downloads.
- Historical market data is downloaded using **`yfinance`**.

**Directory Creation & Data Saving:**

- The 'data' directory is created if absent.
- Data is pickled and saved if not empty.

**Return Statement:**

- The function returns the downloaded data (DataFrame containing adjusted close prices).

## Cell 3 (get_data):

The **get_data** function serves as something like a wrapper function for fetch_data; it checks if the needed data has been previously downloaded or not. If so, it reads the data from the according file; otherwise, it fetches the data.

In this manner, we achieve the functionality of downloading data only once.

Cell 4 (make_combo) :

**Input Parameters:**

- **`capital`**: The initial budget or amount of money available for investment.
- **`stock_data`**: Historical data of stock prices stored in a pandas DataFrame.
- **`weights`**: A set of weights representing the proportion of the total investment allocated to each stock.

**Initialization:**

- **`initial_prices`**: Extracts the stock prices at the initial time point from the provided **`stock_data`**.

**Allocation of Capital:**

- **`units`**: Computes the number of units for each stock. It does this by allocating a proportion of the total **`capital`** based on the specified **`weights`**. This calculation involves dividing the product of **`capital`** and **`weights`** by the initial stock prices.

**Portfolio Valuation:**

- **`values`**: Calculates the value of the portfolio over time. It achieves this by performing a dot product between the entire **`stock_data`** (representing stock prices over time) and the computed **`units`** for each stock.

**Output:**

- The function returns a pandas Series (**`values`**) representing the portfolio values at each time point.

Cell 5 (evaluation metrics) :

## net_profit Function:

**Purpose:**

- Computes the net profit of a portfolio.

**Input:**

- **`values`**: A pandas Series representing the portfolio values over time.
- **`verbose`** (optional): If **`True`**, prints the net profit and percentage return.

**Calculations:**

- Calculates the profit as the difference between the final and initial portfolio values.
- Computes the percentage return based on the initial value.

**Output:**

- Returns the net profit.

## sharpe_ratio Function:

**Purpose:**

- Calculates the Sharpe Ratio, a measure of risk-adjusted return.

**Input:**

- **`values`**: A pandas Series representing the portfolio values over time.
- **`verbose`** (optional): If **`True`**, prints the calculated Sharpe Ratio.

**Calculations:**

- Computes daily returns using percentage changes.
- Calculates the Sharpe Ratio using the formula:

$$\frac{(\textbf{average portfolio return} - \textbf{risk\_free rate})}{\textbf{standard deviation of returns}}$$

- Converts the daily ratio to an annualized ratio.

**Output:**

- Returns the annualized Sharpe Ratio.

## sortino_ratio Function:

**Purpose:**

- Computes the Sortino Ratio, a risk-adjusted return measure that considers only downside risk.

**Input:**

- **`values`**: A pandas Series representing the portfolio values over time.
- **`verbose`** (optional): If **`True`**, prints the calculated Sortino Ratio.

**Calculations:**

- Computes daily returns using percentage changes.
- Calculates the Sortino Ratio using the formula:

$$\frac{(\textbf{average portfolio return} - \textbf{risk\_free rate})}{\textbf{downside standard deviation}}$$

- Converts the daily ratio to an annualized ratio.

**Output:**

- Returns the annualized Sortino Ratio.

Cell 6 (test_weights):

**Purpose:**

- This function serves as a wrapper function around the calling of the functions for each one of the evaluation metrics.

**Input:**

- **`data`**: The DataFrame of historical data over which the weights are going to be combined and tested.
- **`weights`**: The specified combination of capital portion weights allocated to each stock.
- **`start/end`**: The starting and ending time of the tested section.
- **`title`** (optional): A title that can be specified to be printed while printing the test information.

**Operations:**

- Makes the combined portfolio, using the make_combo function defined earlier and using the given weights and data.
- Prints the header info of the test, including the title, the time interval and the weights being tested.
- Proceeds with calling the functions for all of the evaluation metrics in ***verbose*** mode, in order to make them print their calculated information.

Cell 7 (optimize_weights):

**Purpose:**

- This function aims to find the optimal setting of weights for the given data over a specific period of time. This task is performed through the SLSQP method of SciPy package.

**Input:**

- **`data`**: The DataFrame of historical data on which we are trying to find the optimal weights.

- **`verbose`** (optional): If **`True`**, prints the calculated optimal weights while running.

**Operations:**

- Defines a nested function named optimize_target, which is used to reduce repeated code; this function performs the actions needed to achieve optimal weights for a certain metric (net_profit or ...). We will analyze it further in the next section.

- Defines three nested objective functions, one for each metric. These functions are going to be used for the minimize function for the optimization process.

- Finally, it calls the defined optimize_target function, using each of the defined objective functions, in order to optimize the weights according to each metric.

**Output:**

- Returns the optimized weights for each of the metrics.

Cell 7 contd. (optimize_target):

**Purpose:**

- As stated in the previous section, this function is a nested function that is needed in order to perform the operations of optimization for a certain metric.

**Input:**

- **`obj_func`**: The objective function that we are trying to minimize.

- **`title`**: The title for the optimized metric.

- **`verbose`** (optional): If **`True`**, prints the calculated optimal weights while running.

**Operations:**

- Defines constraints that the weights must satisfy.

  These constraints include:

  - The weights need to add up to 1. (the "eq" constraint)
  - Each of the weights needs to be non-negative (the "ineq" constraints)

- Calls the SciPy minimize function, using the SLSQP method, applying the defined constraints.

- Extracts and returns the solved optimized weights. Also prints the weights if ***verbose*** is set.

**Output:**

- Returns the optimized weights for the target metric.

Cell 8 (do_all_for):

**Purpose:**

- This function is the driver function of our code. It utilizes the other functions that we developed to perform the calculations we want and get the result.
- It allows for a generic way of redoing all of the calculations for a different set of symbols, if needed, just by changing the symbols variable and calling the function again.

**Input:**

- **`symbols`**: The set of symbols we want to study.
- **`init_coeffs`** (optional): The initial random non-optimal coefficients we might want to set; if not specified, these will be randomly generated when running the function (hence might be different on each run).

**Operations:**

- Generates random coefficients if ***init_coeffs*** is not specified.
- Tests the initial random weights, on the first period (22/11~23/11), and prints the results.
- Finds the optimized weights for each metric on the first period; then it performs the test with each of the three sets of optimized weights.
- Runs the test again, for a new period of time (23/11/2~23/12/2), once with the initial random weights, and once with the previously optimized weights.
- Finds the new optimized weights for the new period, and performs the test on the new sets of weights also.