بسمه تعالی

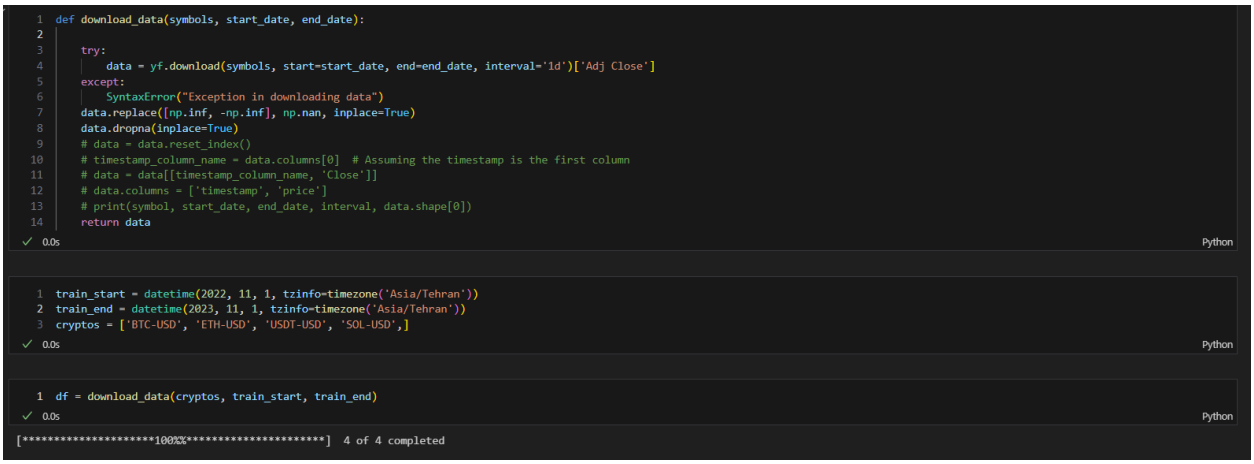
پروژه درس الگوریتم های معاملاتی

دکتر انتظاری

ماهان محمودی – ۹۸۵٤۲٤۳٦

امیرعلی فرازمند – ۹۹۵۲۲۳۲۹

دیتای روزانه را اینگونه از yahoo finance دانلود میکنیم



```python
def download_data(symbols, start_date, end_date):

    try:
        data = yf.download(symbols, start=start_date, end=end_date, interval='1d')['Adj Close']
    except:
        SyntaxError("Exception in downloading data")
    data.replace([np.inf, -np.inf], np.nan, inplace=True)
    data.dropna(inplace=True)
    # data = data.reset_index()
    # timestamp_column_name = data.columns[0]  # Assuming the timestamp is the first column
    # data = data[[timestamp_column_name, 'Close']]
    # data.columns = ['timestamp', 'price']
    # print(symbol, start_date, end_date, interval, data.shape[0])
    return data
```

0.0s Python

```python
train_start = datetime(2022, 11, 1, tzinfo=timezone('Asia/Tehran'))
train_end = datetime(2023, 11, 1, tzinfo=timezone('Asia/Tehran'))
cryptos = ['BTC-USD', 'ETH-USD', 'USDT-USD', 'SOL-USD',]
```

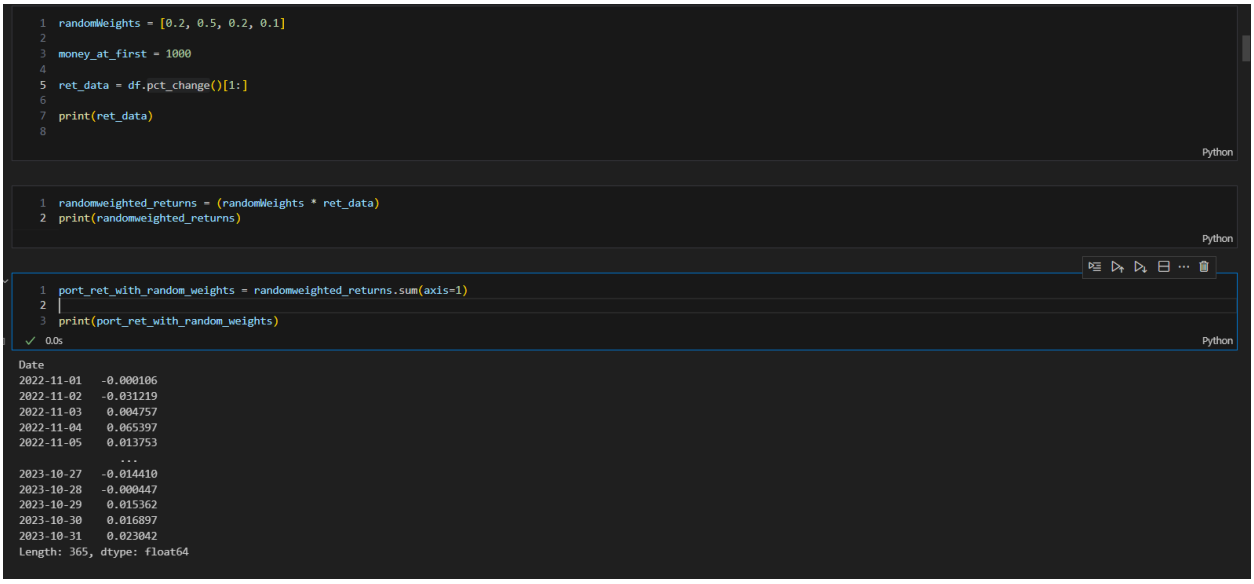
0.0s Python

```python
df = download_data(cryptos, train_start, train_end)
```

0.0s Python

```
[**********100%%**********]  4 of 4 completed
```

یک سری وزن رندوم خودمان به دلخواه تعریف میکنیم. دیتایمان را اینگونه تغیر میدهیم که از روز دوم به بعد برای close هر رمزارز نسبت به روز قبلی اش percent change میگیریم و در وزنش ضرب میکنیم. همچنین برای هر روز حساب میکنیم که در مجموع چند درصد صود یا ضرر کردیم.



```python
randomWeights = [0.2, 0.5, 0.2, 0.1]

money_at_first = 1000

ret_data = df.pct_change()[1:]

print(ret_data)
```

Python

```python
randomweighted_returns = (randomWeights * ret_data)
print(randomweighted_returns)
```

Python

```python
port_ret_with_random_weights = randomweighted_returns.sum(axis=1)

print(port_ret_with_random_weights)
```

0.0s Python

```
Date
2022-11-01   -0.000106
2022-11-02   -0.031219
2022-11-03    0.004757
2022-11-04    0.065397
2022-11-05    0.013753
                ...
2023-10-27   -0.014410
2023-10-28   -0.000447
2023-10-29    0.015362
2023-10-30    0.016897
2023-10-31    0.023042
Length: 365, dtype: float64
```

معیار های گفته شده را به ازای همان وزن های رندوم حساب میکنیم تا بعدا با معیار هایی که بدست می آیند مقایسه کنیم.

Random Weight statistics

+ Code + Markdown

```python
expected_port_return = np.mean(port_ret_with_random_weights)
annual_risk_free_ratio = 0.04
daily_risk_free_ratio =  (1+annual_risk_free_ratio) ** (1/365) -1
std_of_returns = np.std(port_ret_with_random_weights)
initialCapital = 1000
randomWeightedSharpRatio = np.sqrt(365) * ((expected_port_return - daily_risk_free_ratio) / std_of_returns)
randomWeightedSortinoRatio = np.sqrt(365) * sortinoRate(port_ret_with_random_weights , 0)
port_cumulative_returns  = np.cumprod(1+port_ret_with_random_weights)
net_profit_over_time = initialCapital * port_cumulative_returns

final_net_profit = net_profit_over_time[-1] - initialCapital
final_net_profit_percent = final_net_profit/initialCapital * 100
print("Random Weight Sharpe Ratio:{:.2f}".format(randomWeightedSharpRatio))
print("Random Weight Sortino Ratio:{:.2f}".format(randomWeightedSortinoRatio))
print("Random Weight Net Profit: {:.4f}$".format(final_net_profit) + " percentage:" + "{:.2f}%".format(final_net_profit_percent))
```

✓ 0.0s Python

```
Random Weight Sharpe Ratio:0.72
Random Weight Sortino Ratio:1.14
Random Weight Net Profit: 320.9268$ percentage:32.09%
```

توابع تعریف شده برای پیدا کردن وزن های بهینه.

```python
def calculateRequirements(weights:list):
    weighted_rets = ret_data * weights
    port_ret = weighted_rets.sum(axis=1)
    expected_optimized_port_return = np.mean(port_ret)
    stdDevOfRets = np.std(port_ret)
    op_port_cumulative_returns  = np.cumprod(1+port_ret)

    return [expected_optimized_port_return , stdDevOfRets , port_ret , op_port_cumulative_returns]

def calcSharpe(expRet , stdDevRet):
    sharpe = np.sqrt(365) * ((expRet - daily_risk_free_ratio) / stdDevRet)
    return sharpe

def calcSortino(portRets):
    sortino = np.sqrt(365) * sortinoRate(portRets , 0)
    return sortino
def calcNetProfit(op_port_cumulative_returns):
    netPOverTime  = initialCapital * op_port_cumulative_returns
    final_net_p = netPOverTime[-1] - initialCapital
    return final_net_p
def get_reverse_sharpeRatio(weights:list):
    op_expectedPortfolioReturn  , op_stdDevRet , port_rets , op_port_cumulative_returns = calculateRequirements(weights=weights)
    op_sharpe = calcSharpe(op_expectedPortfolioReturn , op_stdDevRet)
    reverseSharpe = 1/(1+op_sharpe)
    return reverseSharpe
def get_reverse_sortinoRatio(weights:list):
    op_expectedPortfolioReturn  , op_stdDevRet, port_rets , op_port_cumulative_returns = calculateRequirements(weights=weights)
    op_sortino = calcSortino(port_rets)
    reverseSortino = 2/(1+op_sortino)
    return reverseSortino

def get_reverse_net_profit(weights:list):
    op_expectedPortfolioReturn  , op_stdDevRet, port_rets , op_port_cumulative_returns = calculateRequirements(weights=weights)
    netProfit = calcNetProfit(op_port_cumulative_returns=op_port_cumulative_returns)
    reverseNetProfit = 1/netProfit
    return reverseNetProfit
```

```python


def check_sum_constraint(weights:list):
    # checks if sum of weights is 1 or not
    return np.sum(weights) - 1

def mix_weights(sharpe_weights, sortino_weights, net_profit_weights) -> list:
    lenght = len(sharpe_weights)
    mixed_weights = [0 for _ in range(lenght)]
    for i in range(lenght):
        mixed_weights[i] = (sharpe_weights[i]+ sortino_weights[i] + net_profit_weights[i]) / 3
    sum = np.sum(mixed_weights)

    if sum < 1:
        rem = sum -1
        max_val = max(mixed_weights)
        index_max_val = mixed_weights.index(max_val)
        mixed_weights[index_max_val] += rem
    elif sum > 1:
        rem = 1 - sum
        max_val = max(mixed_weights)
        index_max_val = mixed_weights.index(max_val)
        mixed_weights[index_max_val] -= rem

    return mixed_weights
```

در تابع calculateRequirements ما اعدادی مثل انحراف معیار، return پورتفولیو را به ازای وزن های داده شده بر میگردانیم.

ما میخواهیم متریک های sharpe ratio, sortino ratio, net profit را ماکسیمایز کنیم. این کار رااینگونه انجام میدهیم که به کمک متد scipy.optimize.minimize وزن ها را طوری بدست بیاوریم که (1+metric) / 1 مینیمایز شود.

حال وزن های بهینه را برای متریک های مختلف در بازه ی ۱ ساله ی گفته شده بدست می اوریم.

Sharpe ratio

setting constraints and bounds on weights

```
bounds = tuple((0,1) for coin in range(len(cryptos)))


constraints = ({'type': 'eq', 'fun': check_sum_constraint})
initialWeightGuess = len(cryptos) * [1/len(cryptos)]

```

0.0s Python

Optimizing Sharpe on 1 year period

```
optimizedSharpe_1year = minimize(get_reverse_sharpeRatio , initialWeightGuess , method='SLSQP' , bounds=bounds , constraints=constraints)

op_weights_for_sharpe_1year = optimizedSharpe_1year.x
print("OUR RANDOM WEIGHTS: ")
print(randomWeights)
print('-' *100)


print("OPTIMIZED WEIGHTS  SHARPE: ")
print(op_weights_for_sharpe_1year)
print('-' *100)


op_expectedPortfolioReturn  , op_stdDevRet, port_rets , op_port_cumulative_returns = calculateRequirements(weights=op_weights_for_sharpe_1year)
optimizedSharpeRatio_1year = calcSharpe(op_expectedPortfolioReturn , op_stdDevRet)


print("SHARPE WITH RANDOM WEIGHTS:")
print(randomWeightedSharpRatio_1year)
print('-' *100)
print("SHARPE WITH OPTIMIZED WEIGHTS:")
print(optimizedSharpeRatio_1year)
```

0.0s Python

```
OUR RANDOM WEIGHTS:
[0.2, 0.5, 0.2, 0.1]
----------------------------------------------------------------------------------------------------
OPTIMIZED WEIGHTS  SHARPE:
[1.00000000e+00 0.00000000e+00 0.00000000e+00 1.40165657e-15]
----------------------------------------------------------------------------------------------------
SHARPE WITH RANDOM WEIGHTS:
0.7154680730266295
----------------------------------------------------------------------------------------------------
SHARPE WITH OPTIMIZED WEIGHTS:
1.2703947162753841
```

Sortino ratio

Optimizing Sortino for the 1 year period

```python
optimizedSortino_1year = minimize(get_reverse_sortinoRatio , initialWeightGuess , method='SLSQP' , bounds=bounds , constraints=constraints)

op_weights_for_sortino_1year = optimizedSortino_1year.x

print("OUR RANDOM WEIGHTS: ")
print(randomWeights)
print('-' *100)


print("OPTIMIZED WEIGHTS FOR SORTINO: ")
print(op_weights_for_sortino_1year)
print('-' *100)

op_expectedPortfolioReturn  , op_stdDevRet, port_rets , op_port_cumulative_returns = calculateRequirements(weights=op_weights_for_sortino_1year)
optimizedSortinoRatio_1year = calcSortino(portRets=port_rets)

print("SORTINO WITH RANDOM WEIGHTS:")
print(randomWeightedSortinoRatio_1year)
print('-' *100)
print("SORTINO WITH OPTIMIZED WEIGHTS:")
print(optimizedSortinoRatio_1year)
```

[16] ✓ 0.0s Python

```
OUR RANDOM WEIGHTS:
[0.2, 0.5, 0.2, 0.1]
----------------------------------------------------------------------------------------------------
OPTIMIZED WEIGHTS FOR SORTINO:
[7.55675381e-01 4.64905892e-16 6.10622664e-16 2.44324619e-01]
----------------------------------------------------------------------------------------------------
SORTINO WITH RANDOM WEIGHTS:
1.1427977239166411
----------------------------------------------------------------------------------------------------
SORTINO WITH OPTIMIZED WEIGHTS:
2.1265314144318967
```

Net profit

```
OUR RANDOM WEIGHTS:
[0.2, 0.5, 0.2, 0.1]
----------------------------------------------------------------------------------------------------
OPTIMIZED WEIGHTS FOR NET PROFIT:
[6.48920814e-01 0.00000000e+00 3.51079186e-01 1.99493200e-17]
----------------------------------------------------------------------------------------------------
NET Profit WITH RANDOM WEIGHTS:
320.93$ percentage: 32.09%
----------------------------------------------------------------------------------------------------
NET PROFIT WITH OPTIMIZED WEIGHTS:
610.59$ percentage: 61.06%
```

حال برای ۱ماه گفته شده هم همین پروسه را تکرار میکنیم. با این تفاوت که این دفعه متریک ها را هم برای وزن های رندوم، هم برای وزن هایی که در یکسال قبلش بدست آورده ایم(از ترکیب وزن های بهینه) و هم وزن های بهینه ی صرفا همان یکماه که بدست می‌آیند بررسی میکنیم.

Sharpe ratio

```
OUR RANDOM WEIGHTS:
[0.2, 0.5, 0.2, 0.1]
----------------------------------------------------------------------------------------------------
OPTIMIZED WEIGHTS FOR SHARPE:
[0.00000000e+00 5.16277478e-01 4.83722522e-01 1.00613933e-17]
----------------------------------------------------------------------------------------------------
SHARPE WITH RANDOM WEIGHTS:
3.88206854612128
----------------------------------------------------------------------------------------------------
SHARPE WITH OPTIMIZED WEIGHTS ON LAST YEAR:
2.658912994609548
----------------------------------------------------------------------------------------------------
SHARPE WITH OPTIMIZED WEIGHTS ON LAST MONTH:
4.204088624853109
```

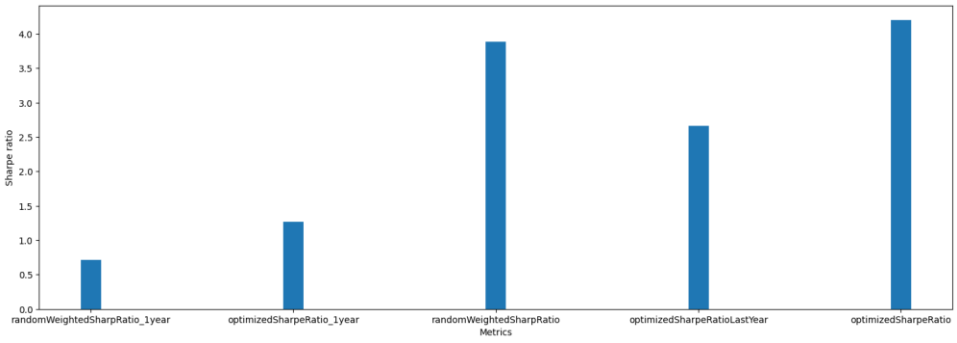
Sortino ratio

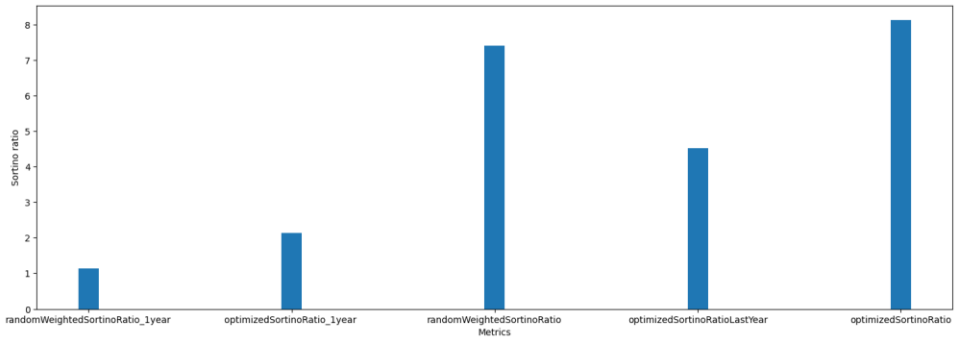
```
OUR RANDOM WEIGHTS:
[0.2, 0.5, 0.2, 0.1]
----------------------------------------------------------------------------------------------------
OPTIMIZED WEIGHTS FOR SORTINO:
[0.00000000e+00 4.00161352e-01 5.99838648e-01 3.61786691e-16]
----------------------------------------------------------------------------------------------------
SORTINO WITH RANDOM WEIGHTS:
7.410485717237132
----------------------------------------------------------------------------------------------------
SORTINO WITH OPTIMIZED WEIGHTS ON LAST YEAR:
4.511040520596667
----------------------------------------------------------------------------------------------------
SORTINO WITH OPTIMIZED WEIGHTS ON LAST MONTH:
8.130256899587566
```

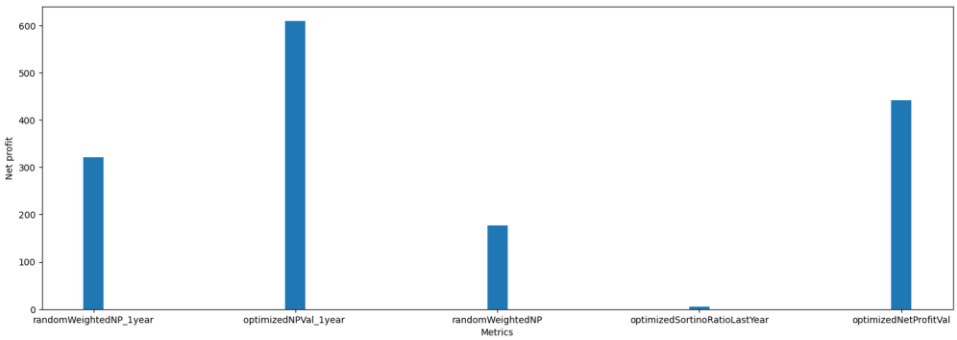
Net profit

```
OUR RANDOM WEIGHTS:
[0.2, 0.5, 0.2, 0.1]
----------------------------------------------------------------------------------------------------
OPTIMIZED WEIGHTS FOR NET PROFIT:
[5.12819878e-17 0.00000000e+00 1.00000000e+00 1.35491691e-17]
----------------------------------------------------------------------------------------------------
NET Profit WITH RANDOM WEIGHTS:
177.24$ percentage: 17.72%
----------------------------------------------------------------------------------------------------
NET PROFIT WITH OPTIMIZED WEIGHTS ON LAST YEAR:
217.13$ percentage: 21.71%
----------------------------------------------------------------------------------------------------
NET PROFIT WITH OPTIMIZED WEIGHTS ON LAST MONTH:
442.40$ percentage: 44.24%
```

در نمودار های زیر به ترتیب از چپ به راست متریک مورد نظر را برای: {(با وزن های رندوم برای سال قبل)، (با وزن های اپتیمایز شده برای سال قبل، روی دیتای سال قبل)، (با وزن های رندوم در ماه تست)، (با وزن های اپتیمایزشده بدست آمده در سال گذشته، بر روی ماه تست)، (اپتیمایز ترین حالت ممکن برای ماه تست) } را مشاهده میکنید.







پ.ن: تابع mix_weights وزن های بهینه ای که بر اساس هر کدام از این متریک ها بدست میاوریم را بهش میدهیم و برای هر رمزارز وزن ها را میانگین میگیرد و در نهایت چک میکند که اگر جمع وزن ها برابر ۱ نباشد از بیشترین وزن مقداری را کم یا زیاد میکند تا مجموع برابر با ۱ شود. در آخر نوت بوک حالتی را گرفتیم که بجای اینکه تک تک متریک ها را با وزن هایی که ماکسیماز کردند آنهارا بررسی کنیم، با یک سری وزن که از این تابع گرفته میشوند بررسی کنیم. این بخش پیاده سازی شده اما ران نشده در نوت بوک، سل هایی هم که برای کار کردن نیاز بهشان داشت هم کامنت شده اند.

منابع:

- پروژه ی مشابه (ریپو)
- Optimizing Portfolio Sharpe Ratio and Weights Using SciPy Bounds, LinearConstraints & Minimize
- Minimizing a function with scipy.optimize
- کد های تمارین
- Chat GPT