006 - Random Portfolio Gen

December 16, 2023

1 #006 Asset Allocation with Random Portfolio Weights

In this code, I'll define a function to generate a random portfolio weights, perform an asset allocation, and then and analyze their returns.

First for a equal weighted portfolio and then for random weights, define a function to concatenate portfolios into a DataFrame and finaly plotly data

1.0.1 libs

Install Libs. (remove comments '#' if need to install the libraries)

[1]:
```
#    !pip install pandas
#    !pip install pandas-datareader
#    !pip install numpy
#    !pip install plotly_express
#    !pip install random
```

[2]:
```
#import Libraries
import pandas as pd
from pandas_datareader import data as pdr
import numpy as np
import random
import plotly.graph_objects as go
```

1.0.2 Functions

functions already defined that we use on this code

[3]:
```
# Define a function using Plotly Express
def plotly_data(df, title):

    # Create figure
    fig = go.Figure()

    # Set title
    fig.update_layout(title_text = title)

    # For loop that plots all stock prices in the pandas dataframe df
```

```
    for i in df.columns[0:]:
        # Add range slider
        #fig.update_layout(xaxis=dict(rangeselector =␣
 ↪dict(buttons=list([dict(count=1, label="1m", step="month",␣
 ↪stepmode="backward"), dict(count=6, label="6m", step="month",␣
 ↪stepmode="backward"), dict(count=1, label="YTD", step="year",␣
 ↪stepmode="todate"), dict(count=1, label="1y", step="year",␣
 ↪stepmode="backward"), dict(step="all")])), rangeslider=dict( visible=True),␣
 ↪type="date"))
        # Add line graph
        fig.add_scatter(x = df.index, y = df[i], name = i)
        # Update Layout
        fig.update_layout({'plot_bgcolor': "white"})
        #fig.update_traces(line_width = 3)
        fig.update_layout(legend=dict(orientation="h",))

    fig.show()

# Define a function using Plotly Express, changes axis y to logarithm scale
def log_plotly_data(df, title):

    # Create figure
    fig = go.Figure()

    # Set title
    fig.update_layout(title_text = title)

    # For loop that plots all stock prices in the pandas dataframe df

    for i in df.columns[0:]:
        # Add range slider
        #fig.update_layout(xaxis=dict(rangeselector =␣
 ↪dict(buttons=list([dict(count=1, label="1m", step="month",␣
 ↪stepmode="backward"), dict(count=6, label="6m", step="month",␣
 ↪stepmode="backward"), dict(count=1, label="YTD", step="year",␣
 ↪stepmode="todate"), dict(count=1, label="1y", step="year",␣
 ↪stepmode="backward"), dict(step="all")])), rangeslider=dict( visible=True),␣
 ↪type="date"))
        # Add line graph
        fig.add_scatter(x = df.index, y = df[i], name = i)
        # Update Layout
        fig.update_layout({'plot_bgcolor': "white"})
        #fig.update_traces(line_width = 3)
        fig.update_layout(legend=dict(orientation="h",))

    #changes y to logarithm scale
```

```
    fig.update_yaxes(type="log")
    fig.show()

# Define a function using Plotly Express, changes axis y to logarithm scale
def plotly_line(df, y, title):

    # Create figure
    fig = go.Figure()
    fig.update_layout(title_text = title)
    fig.add_scatter(x = df.index, y = y)
    # Update Layout
    fig.update_layout({'plot_bgcolor': "white"})
    #fig.update_traces(line_width = 3)
    fig.update_layout(legend=dict(orientation="h",))

    #changes y to logarithm scale
    fig.show()

# Define a function using Plotly Express, changes axis y to logarithm scale
def log_plotly_line(df, y, title):

    # Create figure
    fig = go.Figure()
    fig.update_layout(title_text = title)
    fig.add_scatter(x = df.index, y = y)
    # Update Layout
    fig.update_layout({'plot_bgcolor': "white"})
    #fig.update_traces(line_width = 3)
    fig.update_layout(legend=dict(orientation="h",))

    #changes y to logarithm scale
    fig.update_yaxes(type="log")
    fig.show()
```

[4]:
```
# Function to scale stock prices based on their initial starting price
# The objective of this function is to set all prices to start at a value of 1
def price_scaling(raw_prices_df):
    scaled_prices_df = raw_prices_df.copy()
    for i in raw_prices_df.columns[0:]:
          scaled_prices_df[i] = raw_prices_df[i]/raw_prices_df[i][0]
    return scaled_prices_df
```

1.1 6.1 Equal Weighted Portfolio

[5]:
```
file_name = input('Input the CSV file name: ')
initial_investment = int(input('Input the initial investment: '))
n_runs = int(input('Input the number of simulations: '))
```

```
Input the CSV file name: MAG7
Input the initial investment: 1000000
Input the number of simulations: 5
```

[6]:
```
#read CSV file
Stock_Prices_df = pd.read_csv(file_name)
#The code imports a DataFrame with num index [1,2,3...], this line replace the␣
 ↪colum Date to Index
Stock_Prices_df.set_index(['Date'], inplace = True)
```

[7]:
```
#obtain weights vector
n_assets = len(Stock_Prices_df.columns)
#lock vector to test function
weights = np.ones(n_assets) * 1/n_assets
weights
```

[7]:
```
array([0.14285714, 0.14285714, 0.14285714, 0.14285714, 0.14285714,
       0.14285714, 0.14285714])
```

[8]:
```
#Define a function that performs an Asset Allocation
def asset_allocation(df, initial_investment, weights):
    ''' Performs an asset Allocation for a given DF, initial investment value␣
 ↪and weights'''

    portfolio_df = df.copy()

    # Scale stock prices using the "price_scaling"
    scaled_df = price_scaling(df)

    #enumerate method links Stocks tickers in columns along with a counter␣
 ↪position weight (i), like an index
    for i, stock in enumerate(scaled_df):
        portfolio_df[stock] = weights[i] * scaled_df[stock]  *␣
 ↪initial_investment

    # Sum up all values and place the result in a new column titled "portfolio␣
 ↪value [$]"
    portfolio_df['Total Value [$]'] = portfolio_df.sum(axis = 1, numeric_only =␣
 ↪True)

    # Calculate the portfolio percentage daily return and replace NaNs with␣
 ↪zeros
```

```
    portfolio_df['Daily Return [%]'] = portfolio_df['Total Value [$]'].
 ↪pct_change(1) * 100
    portfolio_df.replace(np.nan, 0, inplace = True)

    return portfolio_df
```

[9]:
```
#Asset Allocation with parameters defined

portfolio_df = asset_allocation(Stock_Prices_df, initial_investment, weights)
Eqw = portfolio_df
portfolio_df.round(2)
```

[9]:

Date	AAPL	AMZN	GOOG	META	MSFT	NVDA	TSLA	Total Value [$]	Daily Return [%]
2018-12-14	142857.14	142857.14	142857.14	142857.14	142857.14	142857.14	142857.14	1000000.00	0.00
2018-12-17	141527.72	136485.65	139351.86	139019.46	138626.56	140057.56	136103.16	971171.97	-2.88
2018-12-18	143366.51	139228.98	141021.57	142460.49	140081.67	143335.15	131653.88	981148.24	1.03
2018-12-19	138894.71	134167.67	140240.18	132127.49	139704.42	135111.95	130067.93	950314.36	-3.14
2018-12-20	135389.74	131094.09	138375.81	132286.15	136767.23	131785.59	123196.76	928895.36	-2.25
…	…	…	…	…	…	…	…	…	…
2023-12-06	691919.20	259382.95	360343.79	314799.41	523268.68	1789366.03	1402575.00	5341655.05	-1.02
2023-12-07	698934.80	263618.65	379590.66	323863.07	526319.22	1832347.28	1421735.40	5446409.08	1.96
2023-12-08	704115.56	264587.83	374628.16	329971.64	530973.02	1868132.27	1428766.72	5501175.19	1.01
2023-12-11	695013.21	261841.80	369309.22	322564.02	526815.78	1833566.32	1404743.05	5413853.40	-1.59
2023-12-12	700517.81	264695.51	366403.01	331429.37	531185.83	1874070.25	1388746.71	5457048.49	0.80

[1257 rows x 9 columns]

[10]:
```
#Plot data:

plotly_line(portfolio_df, portfolio_df['Total Value [$]'], "Portfolio Total␣
 ↪Value")
```

```
log_plotly_line(portfolio_df, portfolio_df['Total Value [$]'], "Portfolio Total␣
↪Value - Log Scale")
plotly_line(portfolio_df, portfolio_df['Daily Return [%]'], "Daily Return [%]")
```

[11]:
```
plotly_data(portfolio_df, "Equal Weighted Porfolio")
log_plotly_data(portfolio_df, "Equal Weighted Porfolio")
```

1.2 6.2 Random Weighted Portfolio

1.2.1 6.2.1 Define a function to generate random weights

[12]:
```
def rand_weights(n):
    ''' Produces n random weights that sum to 1 '''
    k = np.random.rand(n)
    return k / sum(k)
```

[13]:
```
#obtain weights vector
n_assets = len(Stock_Prices_df.columns)

weights = rand_weights(n_assets).round(4)
display(weights)
sum(weights)
```

```
array([0.0133, 0.1626, 0.1925, 0.226 , 0.3021, 0.0629, 0.0406])
```

[13]:
```
1.0
```

1.2.2 6.2.2 Asset Allocation

[14]:
```
#Eqw     --  Equal Weighted </p>
#Rdw_1 -- Random Weighted 1 </p>
#Rdw_2 -- Random Weighted 2 [...]
```

[15]:
```
def random_port_generate(initial_investment, n_runs):
    #obtain weights vector
    n_assets = len(Stock_Prices_df.columns)
    #lock vector to test function
    eq_weights = np.ones(n_assets) * 1/n_assets
    #Asset Allocation with parameters defined
    Eqw_df = asset_allocation(Stock_Prices_df, initial_investment,␣
↪eq_weights)[['Total Value [$]', 'Daily Return [%]']]
    Eqw_df = Eqw_df.rename({'Total Value [$]':'Eqw [$]', 'Daily Return [%]':
↪'Eqw [%]'}, axis="columns")
    All_df = Eqw_df

    # Placeholder to store all weights
    weights_runs = np.zeros((n_runs, n_assets))
```

```
    for i in range(n_runs):
        # Generate random weights
        weights = rand_weights(n_assets)
        # Store the weights
        weights_runs[i,:] = weights
        # Random Asset Allocation
        df = asset_allocation(Stock_Prices_df, initial_investment,␣
↪weights)[['Total Value [$]', 'Daily Return [%]']]
        #rename columns for iterate
        Rdw = df.rename({'Total Value [$]':'Rdw_{}[$]'.format(i), 'Daily Return␣
↪[%]':'Rdw_{}[%]'.format(i)}, axis="columns")

        All_df = pd.merge(All_df, Rdw, on = 'Date')

        #All_df = Eqw_df.join(Rdw)

        print("Simulation Run = {}".format(i))
        print("Weights = {}".format(weights_runs[i].round(3)))
        print('\n')

    return All_df
```

[16]:
```
df = random_port_generate(initial_investment, n_runs)
daily_returns_df = df.iloc[:, 1::2]
total_values_df = df.iloc[:, 0::2]
display(df.round(2))
```

```
Simulation Run = 0
Weights = [0.079 0.184 0.102 0.232 0.093 0.055 0.256]


Simulation Run = 1
Weights = [0.292 0.011 0.06  0.173 0.054 0.133 0.277]


Simulation Run = 2
Weights = [0.161 0.023 0.135 0.26  0.227 0.03  0.163]


Simulation Run = 3
Weights = [0.004 0.187 0.195 0.109 0.224 0.135 0.146]


Simulation Run = 4
Weights = [0.178 0.219 0.045 0.198 0.204 0.15  0.005]
```

```
                Eqw [$]  Eqw [%]    Rdw_0[$]  Rdw_0[%]    Rdw_1[$]  Rdw_1[%]  \
Date
2018-12-14  1000000.00     0.00  1000000.00      0.00  1000000.00      0.00
2018-12-17   971171.97    -2.88   966426.66     -3.36   973379.81     -2.66
2018-12-18   981148.24     1.03   971996.05      0.58   977184.29      0.39
2018-12-19   950314.36    -3.14   939424.81     -3.35   943921.43     -3.40
2018-12-20   928895.36    -2.25   916979.94     -2.39   918395.89     -2.70
…                  …        …          …         …          …         …
2023-12-06  5341655.05    -1.02  5024262.50     -0.47  6551615.59     -0.67
2023-12-07  5446409.08     1.96  5114859.62      1.80  6663627.47      1.71
2023-12-08  5501175.19     1.01  5154780.82      0.78  6728335.71      0.97
2023-12-11  5413853.40    -1.59  5071372.97     -1.62  6617952.78     -1.64
2023-12-12  5457048.49     0.80  5080265.19      0.18  6647266.21      0.44

                Rdw_2[$]  Rdw_2[%]    Rdw_3[$]  Rdw_3[%]    Rdw_4[$]  Rdw_4[%]
Date
2018-12-14  1000000.00      0.00  1000000.00      0.00  1000000.00      0.00
2018-12-17   972134.57     -2.79   967734.13     -3.23   972901.42     -2.71
2018-12-18   980417.02      0.85   977114.46      0.97   990060.82      1.76
2018-12-19   950862.11     -3.01   951418.24     -2.63   952904.31     -3.75
2018-12-20   931704.64     -2.01   930090.47     -2.24   935514.48     -1.82
…                  …         …          …         …          …         …
2023-12-06  4550464.90     -0.48  5040048.22     -1.06  4486420.86     -1.42
2023-12-07  4629599.38      1.74  5144008.52      2.06  4570476.78      1.87
2023-12-08  4665046.11      0.77  5191709.03      0.93  4629740.10      1.30
2023-12-11  4594437.56     -1.51  5111135.59     -1.55  4559151.63     -1.52
2023-12-12  4611750.74      0.38  5146703.50      0.70  4629883.55      1.55

[1257 rows x 12 columns]
```

6.2.2.1 Plotting Data

[17]:
```
plotly_data(total_values_df, "Portfolios Total Value[$]")
log_plotly_data(total_values_df, "Portfolios Total Value[$]")
plotly_data(daily_returns_df, "Portfolio Daily Returns [%]")
```

[18]:
```
import plotly.express as px
# Plot histograms for stocks daily returns using plotly express
fig = px.histogram(daily_returns_df)
fig.update_layout({'plot_bgcolor': "white"})
```