006 - Random Portfolio Gen

December 13, 2023

1 #006 Asset Allocation with Random Portfolio Weights

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

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: '))

## BRL5
```

```
#      PETR4.SA VALE3.SA TAEE11.SA ITSA4.SA WEGE3.SA

## MAG7
#      AAPL AMZN GOOG META MSFT NVDA TSLA

## BRL_top10
#      CRFB3.SA CPLE6.SA ECOR3.SA ITUB4.SA JBSS3.SA RENT3.SA PRIO3.SA SBSP3.SA␣
 ↪VALE3.SA VIVT3.SA
```

```
Input the CSV file name: MAG7
Input the initial investment: 70000
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	10000.00	10000.00	10000.00	10000.00	10000.00	10000.00	10000.00	70000.00	0.00
2018-12-17	9906.94	9554.00	9754.63	9731.36	9703.86	9804.03	9527.22	67982.04	-2.88
2018-12-18	10035.66	9746.03	9871.51	9972.23	9805.72	10033.46	9215.77	68680.38	1.03
2018-12-19	9722.63	9391.74	9816.81	9248.92	9779.31	9457.84	9104.76	66522.01	-3.14
2018-12-20	9477.28	9176.59	9686.31	9260.03	9573.71	9224.99	8623.77	65022.68	-2.25
…	…	…	…	…	…	…	…	…	…
2023-12-06	48434.34	18156.81	25224.07	22035.96	36628.81	125255.62	98180.25	373915.85	-1.02
2023-12-07	48925.44	18453.31	26571.35	22670.42	36842.35	128264.31	99521.48	381248.64	1.96
2023-12-08	49288.09	18521.15	26223.97	23098.02	37168.11	130769.26	100013.67	385082.26	1.01
2023-12-11	48650.92	18328.93	25851.65	22579.48	36877.10	128349.64	98332.01	378969.74	-1.59
2023-12-12	49036.25	18528.69	25648.21	23200.06	37183.01	131184.92	97212.27	381993.39	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([0.1435, 0.2042, 0.0946, 0.1942, 0.0987, 0.08  , 0.1849])
```

[13]:
```
1.0001
```

1.2.2 6.2.2 Asset Allocation Auto

[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 = [0.098 0.036 0.024 0.067 0.211 0.316 0.247]


Simulation Run = 1
Weights = [0.149 0.078 0.056 0.115 0.198 0.179 0.224]


Simulation Run = 2
Weights = [0.211 0.059 0.08  0.161 0.231 0.18  0.078]


Simulation Run = 3
Weights = [0.212 0.094 0.067 0.115 0.185 0.206 0.122]


Simulation Run = 4
```

```
Weights = [0.148 0.075 0.171 0.223 0.002 0.305 0.076]
```

Date	Eqw [$]	Eqw [%]	Rdw_0[$]	Rdw_0[%]	Rdw_1[$]	Rdw_1[%]	\
2018-12-14	70000.00	0.00	70000.00	0.00	70000.00	0.00	
2018-12-17	67982.04	-2.88	67966.56	-2.90	67947.54	-2.93	
2018-12-18	68680.38	1.03	68356.12	0.57	68366.83	0.62	
2018-12-19	66522.01	-3.14	66198.76	-3.16	66308.68	-3.01	
2018-12-20	65022.68	-2.25	64307.40	-2.86	64560.23	-2.64	
…	…	…	…	…	…	…	
2023-12-06	373915.85	-1.02	553619.59	-1.23	450315.01	-0.95	
2023-12-07	381248.64	1.96	563851.67	1.85	458208.03	1.75	
2023-12-08	385082.26	1.01	571134.59	1.29	463201.13	1.09	
2023-12-11	378969.74	-1.59	561649.39	-1.66	455785.59	-1.60	
2023-12-12	381993.39	0.80	567010.60	0.95	458942.04	0.69	

Date	Rdw_2[$]	Rdw_2[%]	Rdw_3[$]	Rdw_3[%]	Rdw_4[$]	Rdw_4[%]
2018-12-14	70000.00	0.00	70000.00	0.00	70000.00	0.00
2018-12-17	68254.94	-2.49	68168.54	-2.62	68281.69	-2.45
2018-12-18	69145.43	1.30	68929.07	1.12	69356.84	1.57
2018-12-19	66862.62	-3.30	66667.35	-3.28	66363.61	-4.32
2018-12-20	65463.02	-2.09	65097.16	-2.36	65101.09	-1.90
…	…	…	…	…	…	…
2023-12-06	388352.90	-1.23	425177.83	-1.21	444582.05	-1.52
2023-12-07	395535.16	1.85	433000.77	1.84	454993.12	2.34
2023-12-08	400330.63	1.21	438217.41	1.20	461271.37	1.38
2023-12-11	394085.36	-1.56	431252.45	-1.59	453187.98	-1.75
2023-12-12	398772.79	1.19	435881.33	1.07	459878.07	1.48

[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"})
```

1.2.3 6.2.3 Asset Allocation Manual

[19]:
```
#Random Asset Allocation 01
weights_1 = rand_weights(len(Stock_Prices_df.columns)).round(4)

print("Value Invested: $ {}".format(initial_investment))
print("Number of Stocks: {}".format(len(Stock_Prices_df.columns)))
print("Weights: {}".format(weights_1))

Rdw_1 = asset_allocation(Stock_Prices_df, initial_investment, weights_1)
Rdw_1.round(2)
```

```
Value Invested: $ 70000
Number of Stocks: 7
Weights: [0.0411 0.4062 0.011  0.2606 0.2414 0.0066 0.033 ]
```

[19]:

Date	AAPL	AMZN	GOOG	META	MSFT	NVDA	TSLA	Total Value [$]	Daily Return [%]
2018-12-14	2877.00	28434.00	770.00	18242.00	16898.00	462.00	2310.00	69993.00	0.00
2018-12-17	2850.23	27165.83	751.11	17751.95	16397.58	452.95	2200.79	67570.43	-3.46
2018-12-18	2887.26	27711.86	760.11	18191.35	16569.70	463.55	2128.84	68712.66	1.69
2018-12-19	2797.20	26704.47	755.89	16871.89	16525.08	436.95	2103.20	66194.68	-3.66
2018-12-20	2726.61	26092.71	745.85	16892.15	16177.65	426.19	1992.09	65053.25	-1.72
…	…	…	…	…	…	…	…	…	…
2023-12-06	13934.56	51627.06	1942.25	40198.00	61895.36	5786.81	22679.64	198063.68	-0.87
2023-12-07	14075.85	52470.13	2045.99	41355.37	62256.20	5925.81	22989.46	201118.81	1.54
2023-12-08	14180.18	52663.03	2019.25	42135.40	62806.67	6041.54	23103.16	202949.23	0.91
2023-12-11	13996.87	52116.47	1990.58	41189.49	62314.93	5929.75	22714.70	200252.79	-1.33
2023-12-12	14107.73	52684.47	1974.91	42321.54	62831.85	6060.74	22456.03	202437.27	1.09

[1257 rows x 9 columns]

[20]:
```
#Random Asset Allocation 02
weights_2 = rand_weights(len(Stock_Prices_df.columns)).round(4)
```

```
print("Value Invested: $ {}".format(initial_investment))
print("Number of Stocks: {}".format(len(Stock_Prices_df.columns)))
print("Weights: {}".format(weights_2))

Rdw_2 = asset_allocation(Stock_Prices_df, initial_investment, weights_2)
Rdw_2.round(2)
```

```
Value Invested: $ 70000
Number of Stocks: 7
Weights: [0.0104 0.2306 0.1875 0.0708 0.0451 0.2168 0.2388]
```

[20]:

Date	AAPL	AMZN	GOOG	META	MSFT	NVDA	TSLA	Total Value [$]	Daily Return [%]
2018-12-14	728.00	16142.00	13125.00	4956.00	3157.00	15176.00	16716.00	70000.00	0.00
2018-12-17	721.23	15422.06	12802.95	4822.86	3063.51	14878.59	15925.70	67636.91	-3.38
2018-12-18	730.60	15732.04	12956.36	4942.24	3095.66	15226.78	15405.08	68088.76	0.67
2018-12-19	707.81	15160.14	12884.57	4583.77	3087.33	14353.21	15219.51	65996.33	-3.07
2018-12-20	689.95	14812.85	12713.28	4589.27	3022.42	13999.85	14415.50	64243.11	-2.66
…	…	…	…	…	…	…	…	…	…
2023-12-06	3526.02	29308.72	33106.59	10921.02	11563.71	190087.93	164118.11	442632.10	-1.09
2023-12-07	3561.77	29787.33	34874.89	11235.46	11631.13	194653.92	166360.10	452104.60	2.14
2023-12-08	3588.17	29896.84	34418.96	11447.38	11733.97	198455.43	167182.85	456723.60	1.02
2023-12-11	3541.79	29586.55	33930.28	11190.39	11642.10	194783.42	164371.79	449046.33	-1.68
2023-12-12	3569.84	29909.00	33663.28	11497.95	11738.68	199086.23	162500.03	451965.00	0.65

[1257 rows x 9 columns]

[21]:
```
#Random Asset Allocation 03
weights_3 = rand_weights(len(Stock_Prices_df.columns)).round(4)

print("Value Invested: $ {}".format(initial_investment))
print("Number of Stocks: {}".format(len(Stock_Prices_df.columns)))
print("Weights: {}".format(weights_3))
```

```
Rdw_3 = asset_allocation(Stock_Prices_df, initial_investment, weights_3)
Rdw_3.round(2)
```

```
Value Invested: $ 70000
Number of Stocks: 7
Weights: [0.0653 0.2018 0.2188 0.0631 0.12   0.1424 0.1886]
```

[21]:

Date	AAPL	AMZN	GOOG	META	MSFT	NVDA	TSLA	Total Value [$]	Daily Return [%]
2018-12-14	4571.00	14126.00	15316.00	4417.00	8400.00	9968.00	13202.00	70000.00	0.00
2018-12-17	4528.46	13495.97	14940.19	4298.34	8151.24	9772.66	12577.84	67764.71	-3.19
2018-12-18	4587.30	13767.24	15119.20	4404.74	8236.80	10001.35	12166.66	68283.30	0.77
2018-12-19	4444.21	13266.77	15035.43	4085.25	8214.62	9427.57	12020.10	66493.95	-2.62
2018-12-20	4332.07	12962.85	14835.55	4090.16	8041.91	9195.47	11385.11	64843.10	-2.48
…	…	…	…	…	…	…	…	…	…
2023-12-06	22139.34	25648.30	38633.18	9733.28	30768.20	124854.80	129617.57	381394.67	-0.97
2023-12-07	22363.82	26067.14	40696.67	10013.52	30947.57	127853.86	131388.26	389330.84	2.08
2023-12-08	22529.59	26162.97	40164.63	10202.39	31221.21	130350.80	132038.05	392669.64	0.86
2023-12-11	22238.34	25891.44	39594.38	9973.36	30976.77	127938.92	129817.92	386431.13	-1.59
2023-12-12	22414.47	26173.62	39282.80	10247.46	31233.73	130765.13	128339.64	388456.84	0.52

[1257 rows x 9 columns]

[22]:
```
#Random Asset Allocation 04
weights_4 = rand_weights(len(Stock_Prices_df.columns)).round(4)

print("Value Invested: $ {}".format(initial_investment))
print("Number of Stocks: {}".format(len(Stock_Prices_df.columns)))
print("Weights: {}".format(weights_4))

Rdw_4 = asset_allocation(Stock_Prices_df, initial_investment, weights_4)
Rdw_4.round(2)
```

```
Value Invested: $ 70000
Number of Stocks: 7
Weights: [0.0541 0.1346 0.1001 0.0212 0.1335 0.3238 0.2328]
```

[22]:

Date	AAPL	AMZN	GOOG	META	MSFT	NVDA	TSLA	Total Value [$]	Daily Return [%]
2018-12-14	3787.00	9422.00	7007.00	1484.00	9345.00	22666.00	16296.00	70007.00	0.00
2018-12-17	3751.76	9001.77	6835.07	1444.13	9068.26	22221.81	15525.56	67848.36	-3.08
2018-12-18	3800.50	9182.71	6916.97	1479.88	9163.44	22741.84	15018.02	68303.36	0.67
2018-12-19	3681.96	8848.89	6878.64	1372.54	9138.76	21437.13	14837.11	66195.04	-3.09
2018-12-20	3589.05	8646.18	6787.20	1374.19	8946.63	20909.37	14053.30	64305.90	-2.85
…	…	…	…	…	…	…	…	…	…
2023-12-06	18342.09	17107.34	17674.50	3270.14	34229.62	283904.39	159994.54	534522.62	-1.30
2023-12-07	18528.06	17386.70	18618.54	3364.29	34429.17	290723.88	162180.20	545230.86	2.00
2023-12-08	18665.40	17450.63	18375.14	3427.75	34733.60	296401.60	162982.28	552036.39	1.25
2023-12-11	18424.11	17269.51	18114.25	3350.79	34461.65	290917.30	160241.85	542779.47	-1.68
2023-12-12	18570.03	17457.73	17971.70	3442.89	34747.52	297343.73	158417.11	547950.71	0.95

[1257 rows x 9 columns]

[23]:
```python
#Random Asset Allocation 02
weights_5 = rand_weights(len(Stock_Prices_df.columns)).round(4)

print("Value Invested: $ {}".format(initial_investment))
print("Number of Stocks: {}".format(len(Stock_Prices_df.columns)))
print("Weights: {}".format(weights_5))

Rdw_5 = asset_allocation(Stock_Prices_df, initial_investment, weights_5)
Rdw_5.round(2)
```

```
Value Invested: $ 70000
Number of Stocks: 7
Weights: [0.1721 0.146  0.1036 0.0297 0.248  0.0698 0.2307]
```

[23]:
```
                AAPL      AMZN      GOOG     META      MSFT      NVDA  \
Date
2018-12-14  12047.00  10220.00   7252.00  2079.00  17360.00   4886.00
2018-12-17  11934.89   9764.18   7074.06  2023.15  16845.90   4790.25
2018-12-18  12089.95   9960.44   7158.82  2073.23  17022.72   4902.35
2018-12-19  11712.85   9598.36   7119.15  1922.85  16976.88   4621.10
2018-12-20  11417.28   9378.47   7024.51  1925.16  16619.95   4507.33
…                …         …         …        …         …         …
2023-12-06  58348.85  18556.26  18292.49  4581.28  63587.61  61199.90
2023-12-07  58940.47  18859.28  19269.54  4713.18  63958.31  62669.94
2023-12-08  59377.36  18928.61  19017.62  4802.08  64523.84  63893.86
2023-12-11  58609.77  18732.16  18747.61  4694.27  64018.65  62711.64
2023-12-12  59073.97  18936.32  18600.08  4823.29  64549.70  64096.95

                TSLA  Total Value [$]  Daily Return [%]
Date
2018-12-14   16149.00         69993.00              0.00
2018-12-17   15385.51         67817.94             -3.11
2018-12-18   14882.55         68090.06              0.40
2018-12-19   14703.27         66654.46             -2.11
2018-12-20   13926.53         64799.24             -2.78
…                 …                …                 …
2023-12-06  158551.29        383117.67             -0.63
2023-12-07  160717.24        389127.96              1.57
2023-12-08  161512.08        392055.45              0.75
2023-12-11  158796.37        386310.48             -1.47
2023-12-12  156988.09        387068.40              0.20

[1257 rows x 9 columns]
```

6.2.3.1 Organize and Plot Data

[24]:
```
rand_value_df = pd.concat([  Eqw['Total Value [$]'],
                           Rdw_1['Total Value [$]'],
                           Rdw_2['Total Value [$]'],
                           Rdw_3['Total Value [$]'],
                           Rdw_4['Total Value [$]'],
                           Rdw_5['Total Value [$]']
                           ], axis=1, join='inner').round(2)
#rand_value_df.to_csv('rand_value_df')
```

[25]:
```
#read CSV file
rand_value_df = pd.read_csv('rand_value')
#The code imports a DataFrame with num index [1,2,3...], this line replace the␣
 ↪colum Date to Index
rand_value_df.set_index(['Date'], inplace = True)

log_plotly_data(rand_value_df, "Total Value [$]")
```

[26]:
```
rand_daily_df = pd.concat([  Eqw['Daily Return [%]'],
                           Rdw_1['Daily Return [%]'],
                           Rdw_2['Daily Return [%]'],
                           Rdw_3['Daily Return [%]'],
                           Rdw_4['Daily Return [%]'],
                           Rdw_5['Daily Return [%]']
                           ], axis=1, join='inner').round(2)
#rand_daily_df.to_csv('rand_daily')
```

[27]:
```
#read CSV file
rand_daily_df = pd.read_csv('rand_daily')
#The code imports a DataFrame with num index [1,2,3...], this line replace the␣
 ↪colum Date to Index
rand_daily_df.set_index(['Date'], inplace = True)

plotly_data(rand_daily_df, "Total Value [$]")
```