Using Monte Carlo Simulations to find the optimal portfolio weights - portfolio with highest Sharpe Ratio

In [2]:
```
# Either Download the data from yfinance. Here we are importing the Adj Closing Price of 10 st
# from a csv file

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns


close_price_df = pd.read_csv("stock_prices.csv")
```

In [3]:
```
close_price_df.head(5)
```

Out[3]:

	Date	AMZN	CAT	DE	EXC	GOOGL	JNJ	JPM	META	PFE	PG
0	1/2/2014	19.898500	69.512543	75.620926	14.121004	27.855856	71.443314	45.640057	54.709999	20.879053	61.941517
1	1/3/2014	19.822001	69.473892	75.956062	13.835152	27.652653	72.086861	45.992889	54.560001	20.920183	61.872295
2	1/6/2014	19.681499	68.561180	75.327690	13.923512	27.960960	72.463608	46.259468	57.200001	20.940746	62.018425
3	1/7/2014	19.901501	68.785461	75.662811	13.996271	28.500000	74.001869	45.726311	57.919998	21.070980	62.618301
4	1/8/2014	20.096001	68.947906	74.850121	13.965082	28.559309	73.899841	46.157543	58.230000	21.214928	61.710812

In [4]:
```
close_price_df.tail()
```

Out[4]:

	Date	AMZN	CAT	DE	EXC	GOOGL	JNJ	JPM	META	PFE	
2252	12/12/2022	90.550003	233.059998	437.049988	42.500000	93.309998	177.839996	134.210007	114.709999	52.160000	152.
2253	12/13/2022	92.489998	235.490005	437.190002	42.540001	95.629997	179.210007	134.080002	120.150002	53.070000	152.
2254	12/14/2022	91.580002	234.479996	438.440002	42.820000	95.070000	179.759995	133.410004	121.589996	54.480000	152.
2255	12/15/2022	88.449997	230.660004	429.790008	42.380001	90.860001	177.490005	130.100006	116.150002	53.610001	151.
2256	12/16/2022	87.860001	232.720001	431.089996	41.930000	90.260002	175.669998	129.289993	119.430000	51.400002	150.

In [5]:
```
# Calculate the percentage daily return for each stock
# We will perform this calculation on all stocks except for the first column which is "Date"
daily_returns_df = close_price_df.iloc[:, 1:].pct_change() * 100
daily_returns_df.replace(np.nan, 0, inplace = True)
daily_returns_df
```

Out[5]:

	AMZN	CAT	DE	EXC	GOOGL	JNJ	JPM	META	PFE	PG
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1	-0.384451	-0.055602	0.443179	-2.024307	-0.729481	0.900780	0.773077	-0.274169	0.196992	-0.111753
2	-0.708814	-1.313748	-0.827284	0.638662	1.114930	0.522629	0.579608	4.838708	0.098293	0.236179
3	1.117807	0.327126	0.444884	0.522567	1.927829	2.122805	-1.152537	1.258737	0.621916	0.967255
4	0.977313	0.236162	-1.074095	-0.222838	0.208102	-0.137872	0.943073	0.535223	0.683156	-1.449240
...	...	...	...	...	...	...	...	...	...	...
2252	1.638800	2.538609	0.515165	2.607441	0.517070	1.194942	1.551152	-1.026749	0.850732	1.027036
2253	2.142457	1.042653	0.032036	0.094120	2.486336	0.770361	-0.096867	4.742396	1.744632	-0.150847
2254	-0.983886	-0.428897	0.285917	0.658201	-0.585588	0.306896	-0.499700	1.198498	2.656868	0.394108
2255	-3.417782	-1.629133	-1.972903	-1.027554	-4.428315	-1.262789	-2.481072	-4.474048	-1.596914	-1.131900
2256	-0.667039	0.893088	0.302470	-1.061823	-0.660355	-1.025414	-0.622608	2.823933	-4.122363	-0.443384

2257 rows × 10 columns

In [6]:
```
# Insert the date column at the start of the Pandas DataFrame (@ index = 0)
daily_returns_df.insert(0, "Date", close_price_df['Date'])
daily_returns_df
```

Out[6]:

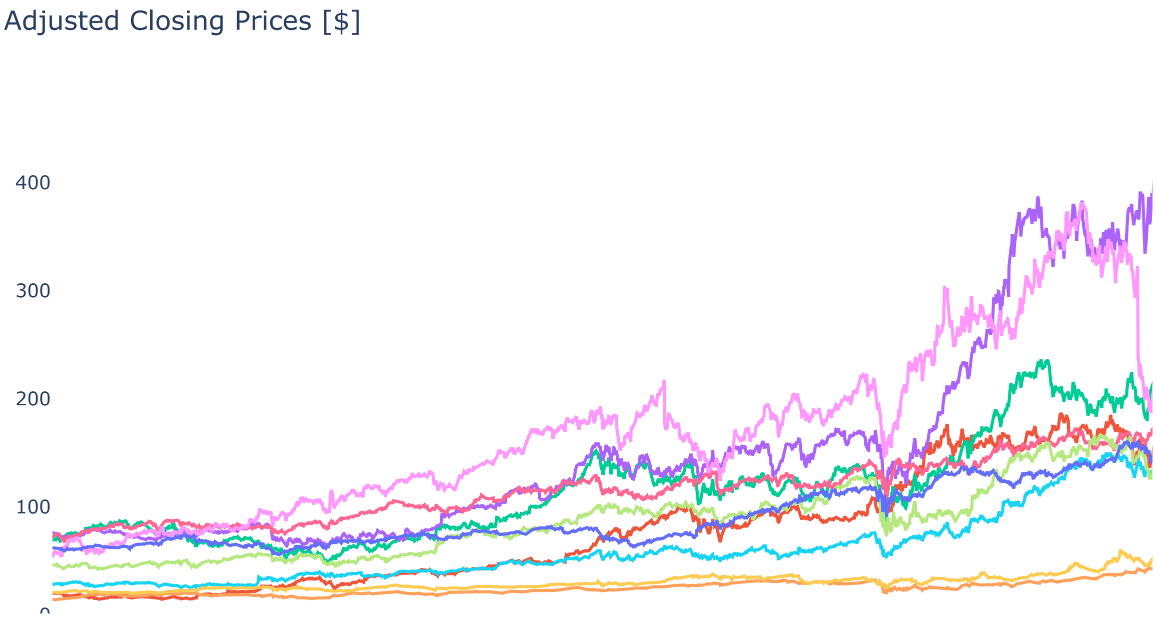
	Date	AMZN	CAT	DE	EXC	GOOGL	JNJ	JPM	META	PFE	PG
0	1/2/2014	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1	1/3/2014	-0.384451	-0.055602	0.443179	-2.024307	-0.729481	0.900780	0.773077	-0.274169	0.196992	-0.111753
2	1/6/2014	-0.708814	-1.313748	-0.827284	0.638662	1.114930	0.522629	0.579608	4.838708	0.098293	0.236179
3	1/7/2014	1.117807	0.327126	0.444884	0.522567	1.927829	2.122805	-1.152537	1.258737	0.621916	0.967255
4	1/8/2014	0.977313	0.236162	-1.074095	-0.222838	0.208102	-0.137872	0.943073	0.535223	0.683156	-1.449240
...	...	...	...	...	...	...	...	...	...	...	...
2252	12/12/2022	1.638800	2.538609	0.515165	2.607441	0.517070	1.194942	1.551152	-1.026749	0.850732	1.027036
2253	12/13/2022	2.142457	1.042653	0.032036	0.094120	2.486336	0.770361	-0.096867	4.742396	1.744632	-0.150847
2254	12/14/2022	-0.983886	-0.428897	0.285917	0.658201	-0.585588	0.306896	-0.499700	1.198498	2.656868	0.394108
2255	12/15/2022	-3.417782	-1.629133	-1.972903	-1.027554	-4.428315	-1.262789	-2.481072	-4.474048	-1.596914	-1.131900
2256	12/16/2022	-0.667039	0.893088	0.302470	-1.061823	-0.660355	-1.025414	-0.622608	2.823933	-4.122363	-0.443384

2257 rows × 11 columns

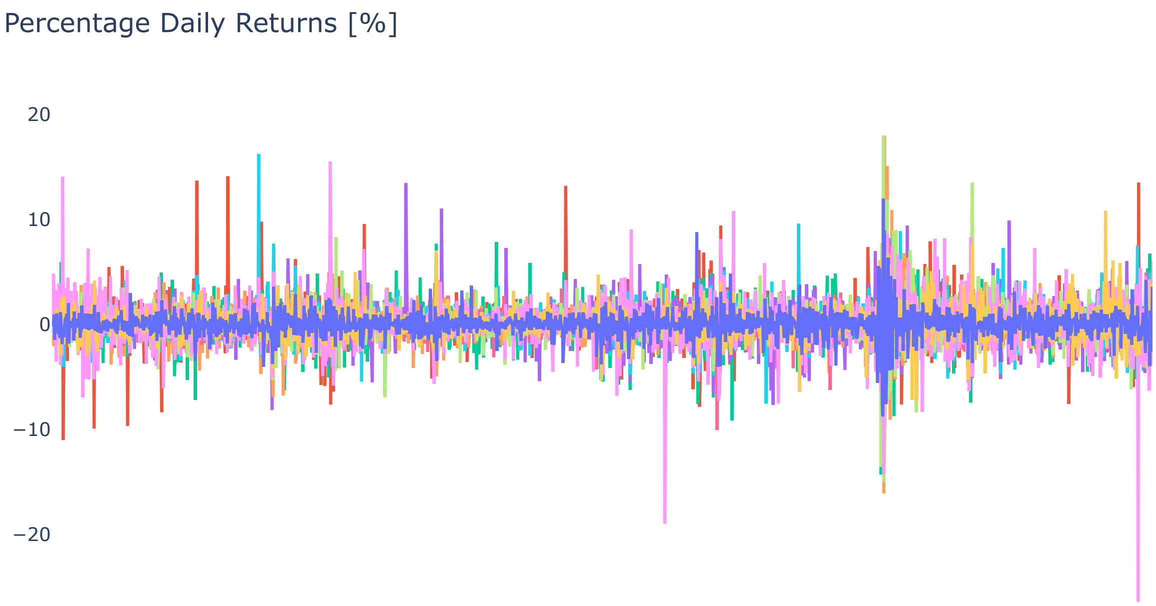
In [7]:
```
#Define a func to plot finacial data for multiple stocks on the same fig
def plot_financial_data(df, title):
    fig = px.line(title=title)

    for i in df.columns[1:]:
        fig.add_scatter(x=df['Date'], y=df[i], name = i)
        fig.update_traces(line_width=2)
        fig.update_layout({'plot_bgcolor': "white"})
    fig.show()
```

In [8]:
```
# Plot stock closing prices
plot_financial_data(close_price_df, 'Adjusted Closing Prices [$]')
```

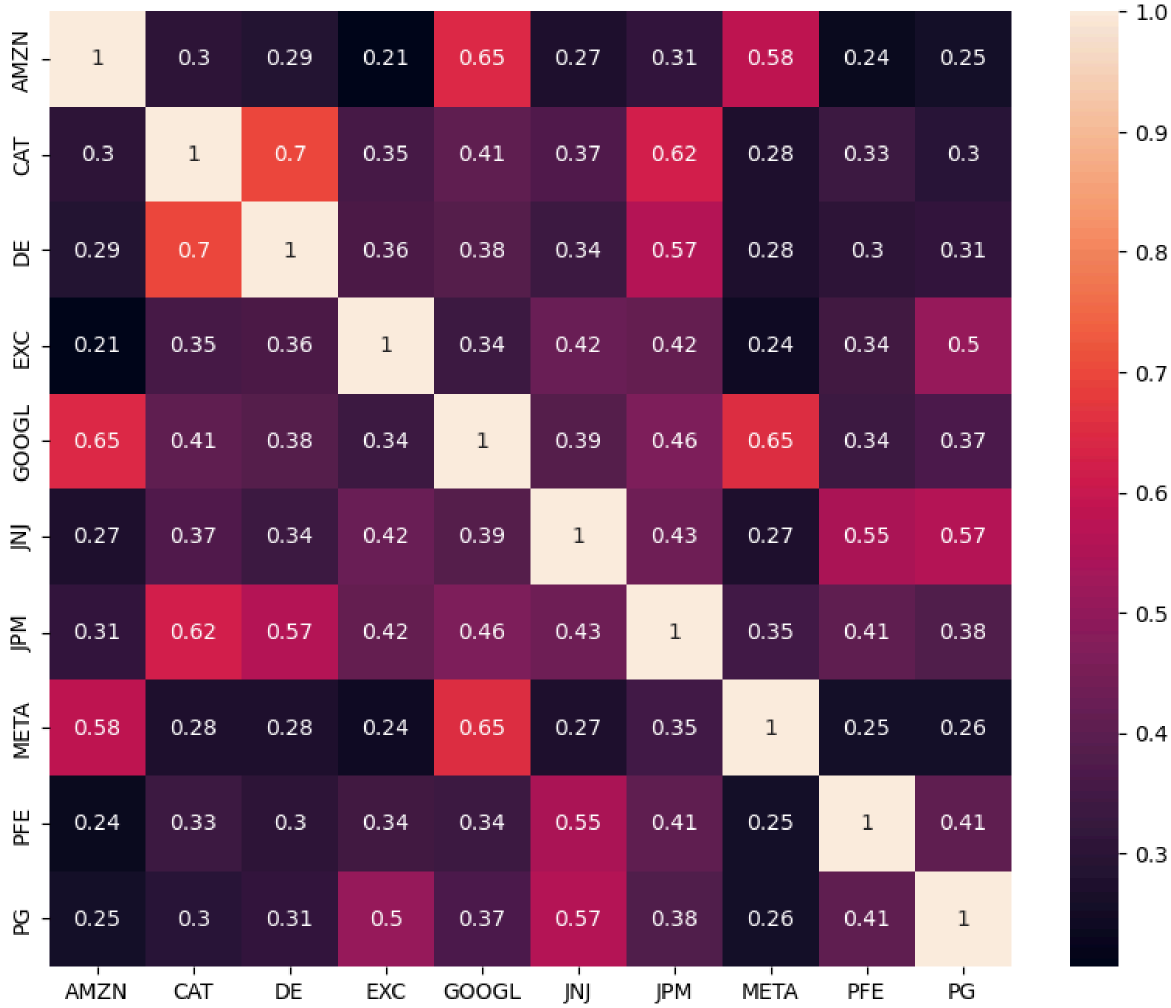


In [9]:
```
# Plot the stocks daily returns
plot_financial_data(daily_returns_df, 'Percentage Daily Returns [%]')
```



In [10]:
```
# Plot histograms for stocks daily returns using plotly express
# Compare META to JNJ daily returns histograms
fig = px.histogram(daily_returns_df.drop(columns = ['Date']))
fig.update_layout({'plot_bgcolor': "white"})
```

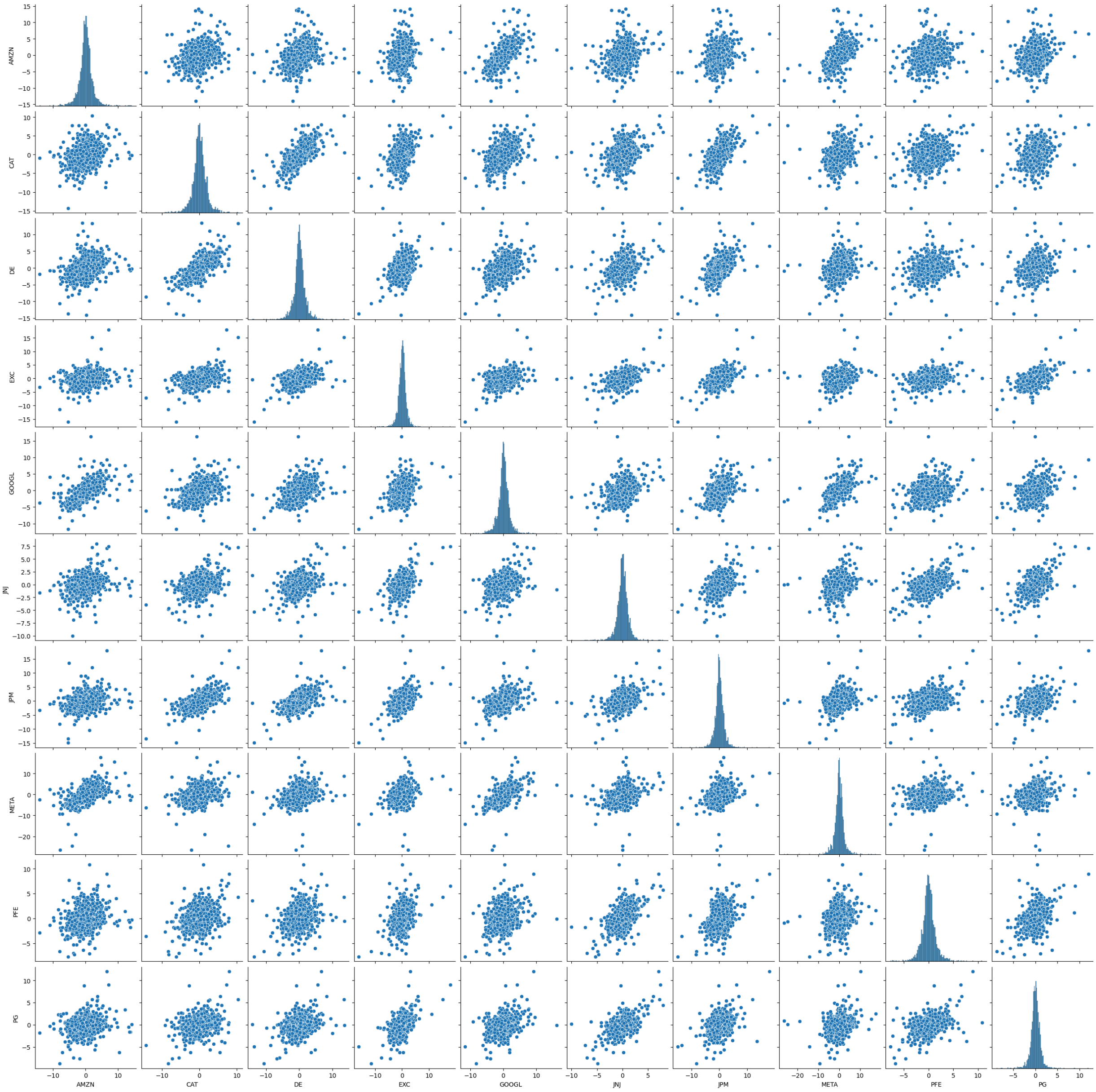
In [11]:
```
# Plot a heatmap showing the correlations between daily returns
# Strong positive correlations between Catterpillar and John Deere - both into heavy equipment
# META and Google - both into Tech and Cloud Computing
plt.figure(figsize = (10, 8))
sns.heatmap(daily_returns_df.drop(columns = ['Date']).corr(), annot = True);
```



In [12]:
```
# Plot the Pairplot between stocks daily returns
sns.pairplot(daily_returns_df);
```

```
/Users/murtazawani/anaconda3/lib/python3.11/site-packages/seaborn/axisgrid.py:118: UserWarnin
g:

The figure layout has changed to tight
```



In [13]:
```
# 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[1:]:
          scaled_prices_df[i] = raw_prices_df[i]/raw_prices_df[i][0]
    return scaled_prices_df
```

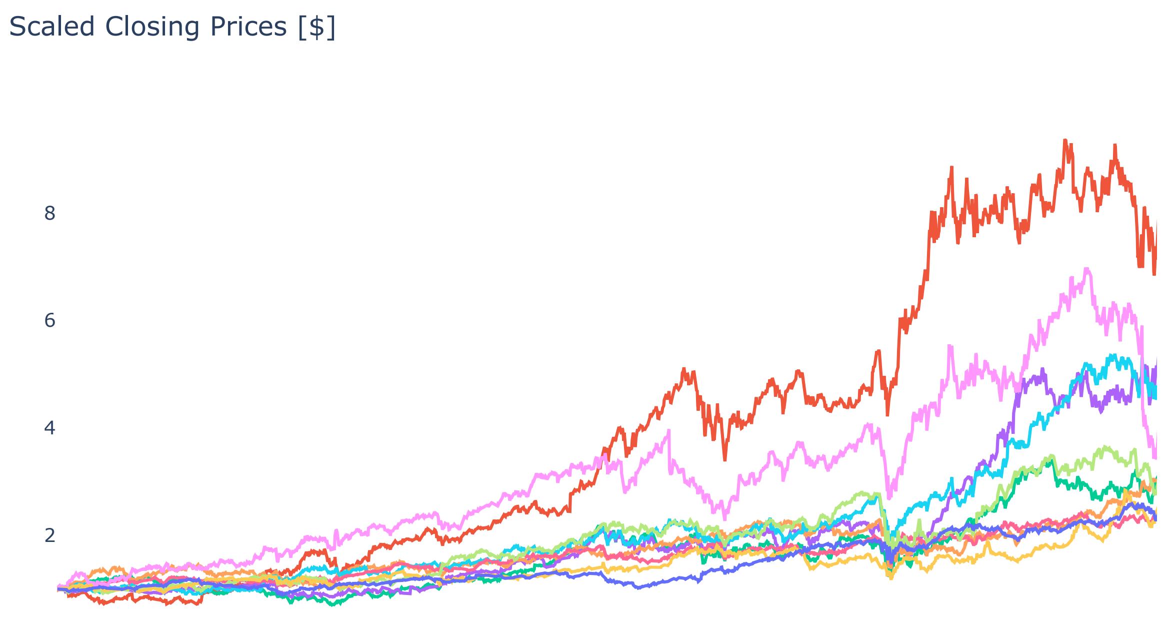
In [14]:
```
price_scaling(close_price_df)
```

Out[14]:

	Date	AMZN	CAT	DE	EXC	GOOGL	JNJ	JPM	META	PFE	PG
0	1/2/2014	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
1	1/3/2014	0.996155	0.999444	1.004432	0.979757	0.992705	1.009008	1.007731	0.997258	1.001970	0.998882
2	1/6/2014	0.989095	0.986314	0.996122	0.986014	1.003773	1.014281	1.013572	1.045513	1.002955	1.001242
3	1/7/2014	1.000151	0.989540	1.000554	0.991167	1.023124	1.035812	1.001890	1.058673	1.009192	1.010926
4	1/8/2014	1.009925	0.991877	0.989807	0.988958	1.025253	1.034384	1.011338	1.064339	1.016087	0.996275
...	...	...	...	...	...	...	...	...	...	...	...
2252	12/12/2022	4.550594	3.352776	5.779485	3.009701	3.349744	2.489246	2.940619	2.096692	2.498198	2.461515
2253	12/13/2022	4.648089	3.387734	5.781336	3.012534	3.433030	2.508422	2.937770	2.196125	2.541782	2.457802
2254	12/14/2022	4.602357	3.373204	5.797866	3.032362	3.412927	2.516121	2.923090	2.222446	2.609314	2.467489
2255	12/15/2022	4.445058	3.318250	5.683480	3.001203	3.261792	2.484347	2.850566	2.123012	2.567645	2.439559
2256	12/16/2022	4.415408	3.347885	5.700671	2.969336	3.240252	2.458872	2.832818	2.182965	2.461798	2.428743

2257 rows × 11 columns

In [15]:
```
#Plot the scaled closing prices data
plot_financial_data(price_scaling(close_price_df), title="Scaled Closing Prices [$] ")
```



In [16]:
```
# We create an array that holds random portfolio weights
# Note that portfolio weights must add up to 1
import random

def generate_portfolio_weights(n):
    weights = []
    for i in range(n):
        weights.append(random.random())

    # let's make the sum of all weights add up to 1
    weights = weights/np.sum(weights)
    return weights
```

In [17]:
```
# Testing the function out
weights = generate_portfolio_weights(4)
print(weights)
print(sum(weights))
```

```
[0.23926912 0.18451745 0.29859814 0.27761529]
1.0
```

In [18]:
```
# Assume that we have $1,000,000 that we would like to invest in one or more of the selected s
# We create a function that receives the following arguments:
      # (1) Stocks closing prices
      # (2) Random weights
      # (3) Initial investment amount
# The function will return a DataFrame that contains the following:
      # (1) Daily value (position) of each individual stock over the specified time period
      # (2) Total daily value of the portfolio
      # (3) Percentage daily return

def asset_allocation(df, weights, initial_investment):
    portfolio_df = df.copy()

    # Scale stock prices using the "price_scaling" function that we defined earlier (Make them
    scaled_df = price_scaling(df)

    for i, stock in enumerate(scaled_df.columns[1:]):
        portfolio_df[stock] = scaled_df[stock] * weights[i] * initial_investment

    # Sum up all values and place the result in a new column titled "portfolio value [$]"
    # Note that we excluded the date column from this calculation
    portfolio_df['Portfolio Value [$]'] = portfolio_df[portfolio_df != 'Date'].sum(axis = 1, n

    # Calculate the portfolio percentage daily return and replace NaNs with zeros
    portfolio_df['Portfolio Daily Return [%]'] = portfolio_df['Portfolio Value [$]'].pct_chang
    portfolio_df.replace(np.nan, 0, inplace = True)

    return portfolio_df
```

In [19]:
```
weights = generate_portfolio_weights(10)
initial_investment = 1000000
```

In [20]:
```
portfolio_df = asset_allocation(close_price_df, weights, initial_investment)
portfolio_df.round(2)
```

Out[20]:

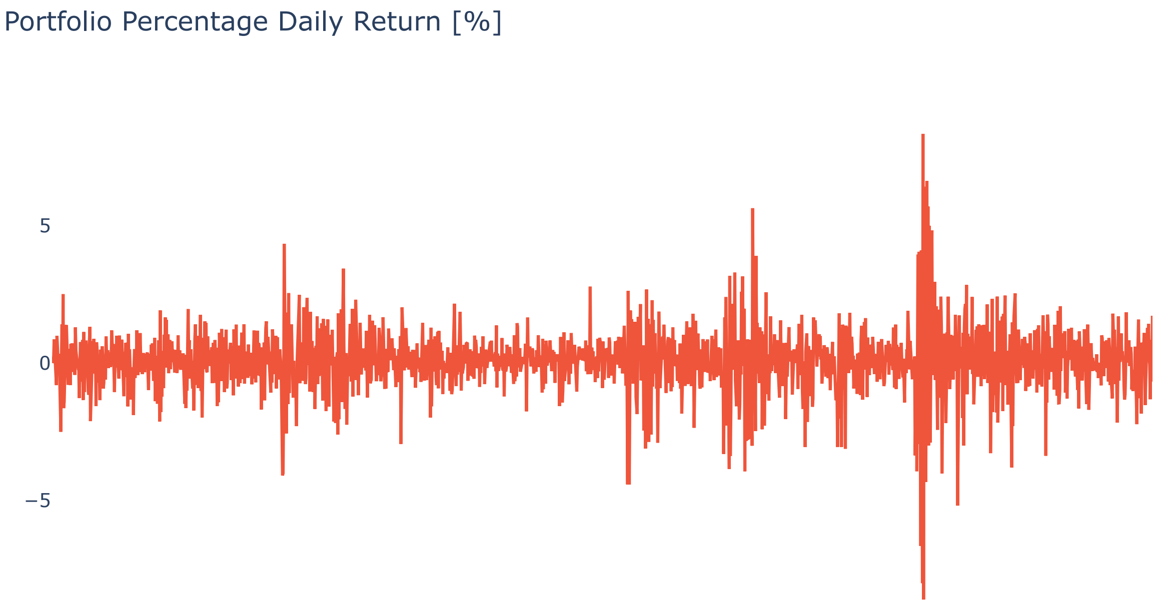
	Date	AMZN	CAT	DE	EXC	GOOGL	JNJ	JPM	META	PFE	PG
0	1/2/2014	115938.82	101345.51	189815.29	2013.36	23785.93	170121.53	57197.43	60908.91	206717.61	72155.61
1	1/3/2014	115493.09	101289.16	190656.52	1972.60	23612.42	171653.95	57639.61	60741.92	207124.83	72074.98
2	1/6/2014	114674.46	99958.48	189079.25	1985.20	23875.68	172551.06	57973.69	63681.04	207328.42	72245.20
3	1/7/2014	115956.30	100285.47	189920.43	1995.57	24335.96	176213.98	57305.52	64482.62	208617.82	72944.00
4	1/8/2014	117089.55	100522.30	187880.50	1991.12	24386.61	175971.03	57845.96	64827.75	210043.01	71886.87
...	...	...	...	...	...	...	...	...	...	...	...
2252	12/12/2022	527590.52	339788.82	1097034.60	6059.60	79676.79	423474.36	168195.83	127707.21	516421.43	177612.16
2253	12/13/2022	538893.92	343331.64	1097386.05	6065.30	81657.82	426736.64	168032.91	133763.59	525431.08	177344.24
2254	12/14/2022	533591.82	341859.10	1100523.66	6105.22	81179.65	428046.28	167193.24	135366.75	539391.09	178043.17
2255	12/15/2022	515354.82	336289.76	1078811.40	6042.49	77584.75	422640.96	163045.06	129310.37	530777.48	176027.90
2256	12/16/2022	511917.20	339293.12	1082074.49	5978.33	77072.42	418307.14	162029.93	132962.01	508896.90	175247.42

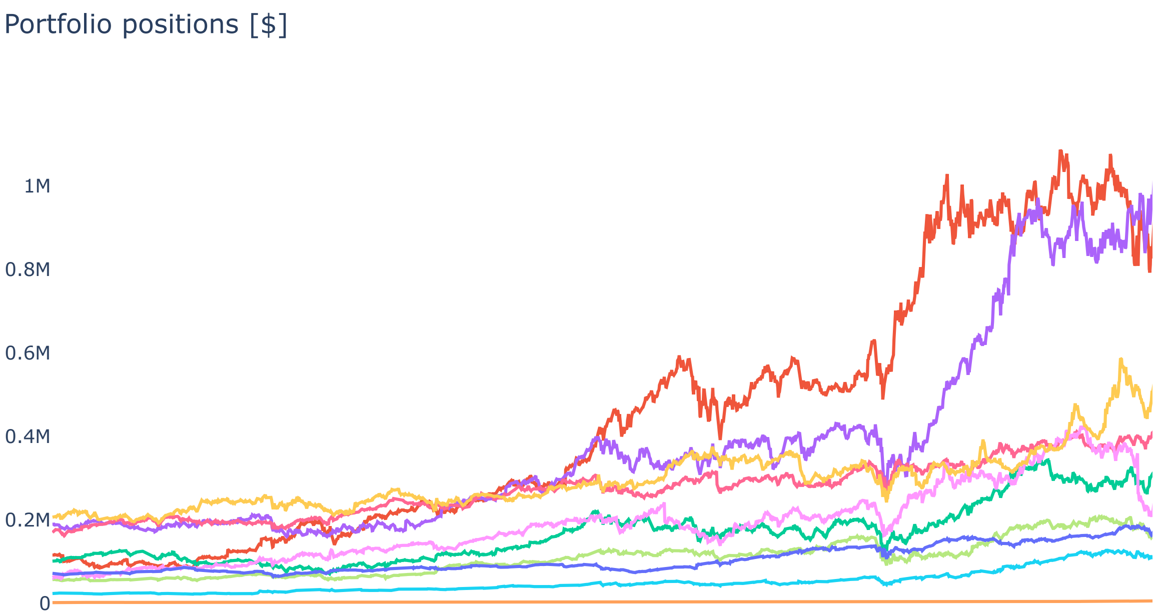
2257 rows × 13 columns

In [21]:
```
# Plot the portfolio percentage daily return
plot_financial_data(portfolio_df[['Date', 'Portfolio Daily Return [%]']], 'Portfolio Percentag

# Plot each stock position in our portfolio over time
# This graph shows how our initial investment in each individual stock grows over time
plot_financial_data(portfolio_df.drop(['Portfolio Value [$]', 'Portfolio Daily Return [%]'], a

# Plot the total daily value of the portfolio (sum of all positions)
plot_financial_data(portfolio_df[['Date', 'Portfolio Value [$]']], 'Total Portfolio Value [$]'
```






Total Portfolio Value [$]
4M
3.5M
3M
2.5M
2M
1.5M
1M

In [22]:
```
# Let's define the simulation engine function
# The function receives:
    # (1) portfolio weights
    # (2) initial investment amount
# The function performs asset allocation and calculates portfolio statistical metrics includin
# The function returns:
    # (1) Expected portfolio return
    # (2) Expected volatility
    # (3) Sharpe ratio
    # (4) Final portfolio value in dollars
    # (5) Return on investment


def simulation_engine(weights, initial_investment):
    # Perform asset allocation using the random weights (sent as arguments to the function)
    portfolio_df = asset_allocation(close_price_df, weights, initial_investment)

    # Calculate the return on the investment
    # Return on investment is calculated using the last final value of the portfolio compared
    return_on_investment = ((portfolio_df['Portfolio Value [$]'][-1:] -
                             portfolio_df['Portfolio Value [$]'][0])/
                             portfolio_df['Portfolio Value [$]'][0]) * 100

    # Daily change of every stock in the portfolio (Note that we dropped the date, portfolio d
    portfolio_daily_return_df = portfolio_df.drop(columns = ['Date', 'Portfolio Value [$]', 'P
    portfolio_daily_return_df = portfolio_daily_return_df.pct_change(1)

    # Portfolio Expected Return formula
    expected_portfolio_return = np.sum(weights * portfolio_daily_return_df.mean() ) * 252

    # Portfolio volatility (risk) formula
    # The risk of an asset is measured using the standard deviation which indicates the disper
    # The risk of a portfolio is not a simple sum of the risks of the individual assets within
    # Portfolio risk must consider correlations between assets within the portfolio which is i
    # The covariance determines the relationship between the movements of two random variables
    # When two stocks move together, they have a positive covariance when they move inversely,

    covariance = portfolio_daily_return_df.cov() * 252
    expected_volatility = np.sqrt(np.dot(weights.T, np.dot(covariance, weights)))

    # Check out the chart for the 10-years U.S. treasury at https://ycharts.com/indicators/10_
    rf = 0.03 # Try to set the risk free rate of return to 1% (assumption)

    # Calculate Sharpe ratio
    sharpe_ratio = (expected_portfolio_return - rf)/expected_volatility
    return expected_portfolio_return, expected_volatility, sharpe_ratio, portfolio_df['Portfol
```

In [23]:
```
# Let's test out the "simulation_engine" function and print out statistical metrics
# Define the initial investment amount
initial_investment = 1000000
portfolio_metrics = simulation_engine(weights, initial_investment)
```

In [24]:
```
print('Expected Portfolio Annual Return = {:.2f}%'.format(portfolio_metrics[0] * 100))
print('Portfolio Standard Deviation (Volatility) = {:.2f}%'.format(portfolio_metrics[1] * 100)
print('Sharpe Ratio = {:.2f}'.format(portfolio_metrics[2]))
print('Portfolio Final Value = ${:.2f}'.format(portfolio_metrics[3]))
print('Return on Investment = {:.2f}%'.format(portfolio_metrics[4]))
```

```
Expected Portfolio Annual Return = 16.58%
Portfolio Standard Deviation (Volatility) = 17.68%
Sharpe Ratio = 0.77
Portfolio Final Value = $3413778.95
Return on Investment = 241.38%
```

In [25]:
```python
#Let's run Monte Carlo simulations by running the above funtion over multiple weights sets

sim_runs = 10000
initial_investment = 1000000
n = 10

# Placeholder to store all weights
weights_runs = np.zeros((sim_runs, n))

# Placeholder to store all expected returns
expected_portfolio_returns_runs = np.zeros(sim_runs)

# Placeholder to store all volatility values
volatility_runs = np.zeros(sim_runs)


# Placeholder to store all Sharpe ratios
sharpe_ratio_runs = np.zeros(sim_runs)

# Placeholder to store all final portfolio values
final_value_runs = np.zeros(sim_runs)


# Placeholder to store all returns on investment
return_on_investment_runs = np.zeros(sim_runs)

for i in range(sim_runs):
    # Generate random weights
    weights = generate_portfolio_weights(n)
    # Store the weights
    weights_runs[i,:] = weights

    # Call "simulation_engine" function and store Sharpe ratio, return and volatility
    # Note that asset allocation is performed using the "asset_allocation" function
    expected_portfolio_returns_runs[i], volatility_runs[i], sharpe_ratio_runs[i], final_value_
    print("Simulation Run = {}".format(i))
    print("Weights = {}, Final Value = ${:.2f}, Sharpe Ratio = {:.2f}".format(weights_runs[i].
    print('\n')
```

```
Weights = [0.059 0.175 0.109 0.066 0.105 0.106 0.046 0.012 0.18  0.165], Final Value = $32
08323.32, Sharpe Ratio = 0.75


Simulation Run = 67
Weights = [0.109 0.18  0.063 0.043 0.203 0.026 0.157 0.116 0.035 0.069], Final Value = $32
40393.42, Sharpe Ratio = 0.69


Simulation Run = 68
Weights = [0.081 0.124 0.088 0.13  0.114 0.073 0.122 0.075 0.138 0.057], Final Value = $31
90622.38, Sharpe Ratio = 0.73


Simulation Run = 69
Weights = [0.061 0.153 0.118 0.139 0.114 0.077 0.097 0.054 0.136 0.051], Final Value = $32
75281.47, Sharpe Ratio = 0.74


Simulation Run = 70
```

In [26]:
```python
#Run with max Sharpe Ratio
sharpe_ratio_runs.argmax()
```

Out[26]: 5211

In [27]:
```python
sharpe_ratio_runs.max()
```

Out[27]: 0.8143491268005124

In [28]:
```
# Obtain the portfolio weights that correspond to the maximum Sharpe ratio (Golden set of weig
weights_runs[sharpe_ratio_runs.argmax(), :]
```

Out[28]:
```
array([0.21183053, 0.12265763, 0.22626394, 0.0743913 , 0.03238587,
       0.03810591, 0.02391695, 0.00549857, 0.09690676, 0.16804253])
```

In [29]:
```
est weights allocation (maximum Sharpe ratio)
iarpe_ratio, optimal_portfolio_final_value, optimal_return_on_investment = simulation_engine(we
```

In [30]:
```
print('Best Portfolio Metrics Based on {} Monte Carlo Simulation Runs:'.format(sim_runs))
print('  - Portfolio Expected Annual Return = {:.02f}%'.format(optimal_portfolio_return * 100)
print('  - Portfolio Standard Deviation (Volatility) = {:.02f}%'.format(optimal_volatility * 1(
print('  - Sharpe Ratio = {:.02f}'.format(optimal_sharpe_ratio))
print('  - Final Value = ${:.02f}'.format(optimal_portfolio_final_value))
print('  - Return on Investment = {:.02f}%'.format(optimal_return_on_investment))
```

```
Best Portfolio Metrics Based on 10000 Monte Carlo Simulation Runs:
  - Portfolio Expected Annual Return = 18.00%
  - Portfolio Standard Deviation (Volatility) = 18.42%
  - Sharpe Ratio = 0.81
  - Final Value = $3781799.27
  - Return on Investment = 278.18%
```

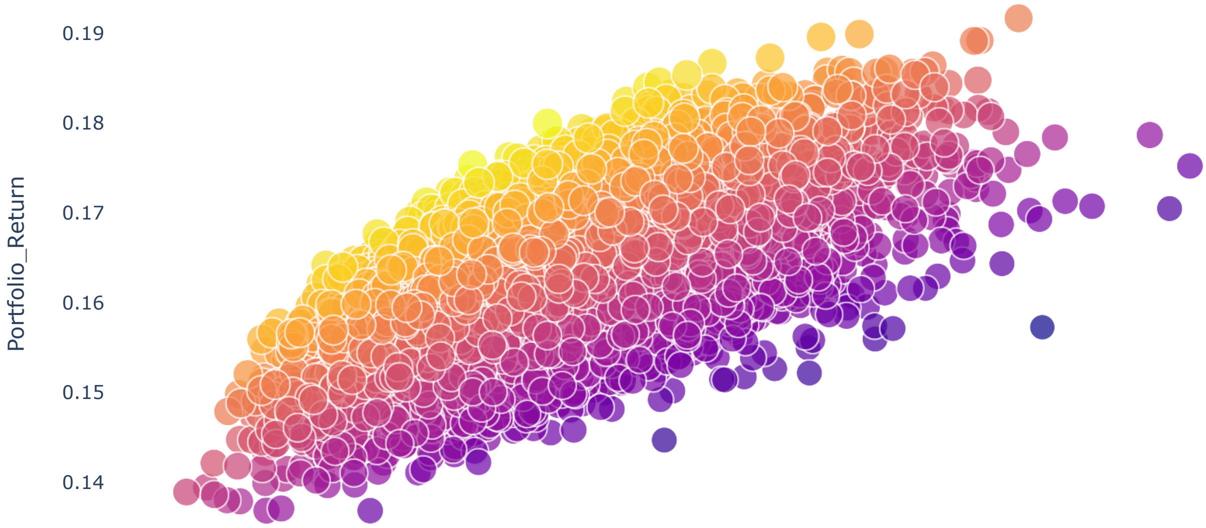
In [31]:
```
# Create a DataFrame that contains volatility, return, and Sharpe ratio for all simualation ru
sim_out_df = pd.DataFrame({'Volatility': volatility_runs.tolist(), 'Portfolio_Return': expecte
sim_out_df
```

Out[31]:

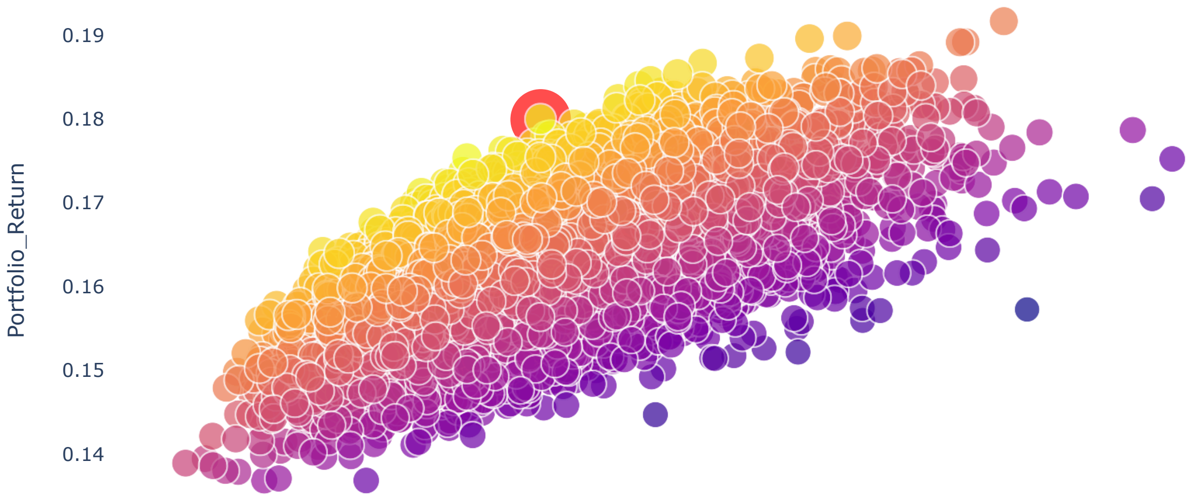
	Volatility	Portfolio_Return	Sharpe_Ratio
0	0.179925	0.158362	0.713417
1	0.188436	0.153359	0.654649
2	0.186065	0.166909	0.735809
3	0.174441	0.157039	0.728263
4	0.204028	0.166594	0.669487
...	...	...	...
9995	0.186843	0.171355	0.756546
9996	0.178487	0.163160	0.746048
9997	0.184988	0.162646	0.717053
9998	0.188797	0.170067	0.741893
9999	0.184139	0.156351	0.686170

10000 rows × 3 columns

In [32]:
```
# Plot volatility vs. return for all simulation runs
# Highlight the volatility and return that corresponds to the highest Sharpe ratio
# This gives us a idea of a Markowitz efficient frontier. A concave down curve which higlights
# highest return for a given level of risk
import plotly.graph_objects as go
fig = px.scatter(sim_out_df, x = 'Volatility', y = 'Portfolio_Return', color = 'Sharpe_Ratio',
fig.update_layout({'plot_bgcolor': "white"})
fig.show()
```



In [33]:
```
# Let's highlight the point with the highest Sharpe ratio
fig = px.scatter(sim_out_df, x = 'Volatility', y = 'Portfolio_Return', color = 'Sharpe_Ratio',
fig.add_trace(go.Scatter(x = [optimal_volatility], y = [optimal_portfolio_return], mode = 'mar
fig.update_layout(coloraxis_colorbar = dict(y = 0.7, dtick = 5))
fig.update_layout({'plot_bgcolor': "white"})
fig.show()
```



In [ ]: