How to outperform the market ? - An introduction to risk management and quantitative finance.

"I want to outperform the market, should I invest in an ETF or select my own portfolio?"

Introduction

For over 20 years, the stock market has continuously increased. The S&P 500, the stock market index based on the 500 largest U.S. companies, has on average increased by 10% per year over these last 20 years, and 7.5% when taking inflation into account. However, when looking at investment funds, 68% of professional funds focusing on the stock market have not beaten their benchmark over 5 years (Khanna, 2023). The benchmark here is a reference yield for comparing the performance of a stock or a strategy against a stock market index, such as the S&P 500. So, how can this be explained? And is there a way for an individual investor to surpass their benchmark and thereby generate a positive alpha, or in other words, exceed the return of their benchmark by beating the market (Chen, 2023).

Background and Context

Outperforming the market is not necessarily a goal for asset managers. Hedge Funds, for example, specialised in equities (equities fund), aim more at having a portfolio that is uncorrelated with the market and often seek a market-neutral strategy (BlackRock, 2023) that is less dependent on market conditions (Guzun, 2023). However, in our case, we aim to beat the market. The situation is as follows: I have $100, I want to invest it all at once, and I have the choice between the **Invesco QQQ**, which is an ETF, Exchange-Traded Fund, replicating the performance of the NASDAQ-100, an index of 100 American companies. It is known for its strong growth and the index includes values like Apple, Microsoft, Nvidia, or Tesla (Invesco, 2023). But I also have the choice of investing directly in stocks that I choose. My goal here is to beat our benchmark, which is the MSCI All Country World Index, or **ACWI**, which accounts for stocks from 23 developed countries and 24 emerging countries (IShares, 2024). We have chosen the **ACWI** as our Benchmark because it encompasses a lot more countries than the S&P500, which is often used as a benchmark.

Introduction to our investment strategy

I have chosen 10 companies, following a particular strategy. I chose to invest in undervalued companies with strong growth, and which have low debt levels. In other words, I selected companies that had a relatively low Price-Earnings Ratio, indicating that the company might be undervalued and could increase in value over time. Additionally, I considered the Return on Invested Capital (ROIC) of the company; if it's positive, it means that the company generates returns on its invested capital. I selected companies with a significant ROIC. Lastly, my selection criteria were that the companies must have low debt levels. Please note that my strategy is personal and that my algorithm and approach can be completely replicated with another strategy or with other assets. I used finviz.com to screen the stocks according to my criteria.
My list of stocks is as follows *(click on each symbol to see more details)*.

- $XOM
- $SU
- $EBAY
- $PM
- $ACLS
- $RYAAY
- $CSCO
- $UMC
- $TRI
- $MMT

With our other investment choice, the $QQQ, (the NASDAQ-100) and our benchmark $ACWI, (All Country World)

Introduction of the scenario

Our Bayesian game will be modeled using a Monte Carlo simulation. This is an algorithm that allows for the prediction of possible outcomes based on uncertain events. We will use the Yahoo Finance API and for each stock, for **QQQ** and for **ACWI** our benchmark. We will calculate the volatility and the average annual return over a period of 10 years. We will not take into account inflation, brokerage fees, or the reinvestment of stock dividends as this would be too complex to establish and could skew our results. We will use these results to perform a Monte Carlo simulation, and then we will conduct a Bayesian game using the estimated returns of our assets for the payoffs. We will conduct 10,000 simulations, and the algorithm will give us the different quartiles of distribution.

Game Presentation

Objective: My goal is to outperform the market (**ACWI**). I have the choice between investing in **QQQ** or investing in my portfolio of 10 stocks (we call it **MPort**, which is the arithmetic mean of our 10 selected stocks).
Players: Myself and the ACWI.
Actions : Me: Buy $100 of **QQQ** or $100 of **MPort** (average of the 10 selected stocks).
State of nature : the **ACWI**, the market can be bullish or bearish. It is assumed that the market cannot be predicted, so there is a 50% chance that the market will be bullish and a 50% chance it will be bearish.
Actions : ACWI **:**

- If the state of nature is bullish, the **ACWI** can achieve a maximum annual return or be in the top 75% of its annual returns.
- If the state of nature is bearish, the **ACWI** can achieve its minimum annual return or be in the lowest 25% of its annual returns.

Here, the state of nature is the **ACWI** player, representing the market, which can act according to two different states: Bullish and Bearish. This represents our uncertainty.
The payoffs have all been given according to our algorithm, and thus the Monte Carlo simulation has provided us with all our values. *(Left table).* On the right, we have our Bayesian game when we invest $100 taking into account the payoffs from our Monte Carlo simulation table on the left, which are the percentage returns.

	Top 75%	Bottom 25%	Maximum	Minimum
ACWI	0.22159	-0.02239	1.15407	-0.39907
QQQ	0.38755	0.04357	2.05102	-0.49654
MPort	0.41642	-0.08096	2.56815	-0.65640

Nature

Bullish Case - 50%

Me \ ACWI	TOP 75%	Record +
MPort	**$41.64** ; $22.16	**$256.82** ; $115.41
QQQ	$38.75 ; $22.16	$205.10 ; $115.41

Bearish Case - 50%

Me \ ACWI	BOTTOM 25%	Record -
MPort	-$8.10 ; -$2.24	-$65.64 ; **-$39.91**
QQQ	**$4.36** ; -$2.24	-$49.65 ; **-$39.91**

For the MPort investment, (Our 10 stocks) :

In the bullish case (50% probability), the average of the Top 75% and Record High is (41.64 + 256.82) / 2 = 149.23.
In the bearish case (50% probability), the average of the Bottom 25% and Record Low is (-8.10 - 65.64) / 2 = -36.87.
The expected payoff for **MPort** is then the average of the bullish and bearish cases: (149.23 + (-36.87)) / 2 ≈ **56.18**.

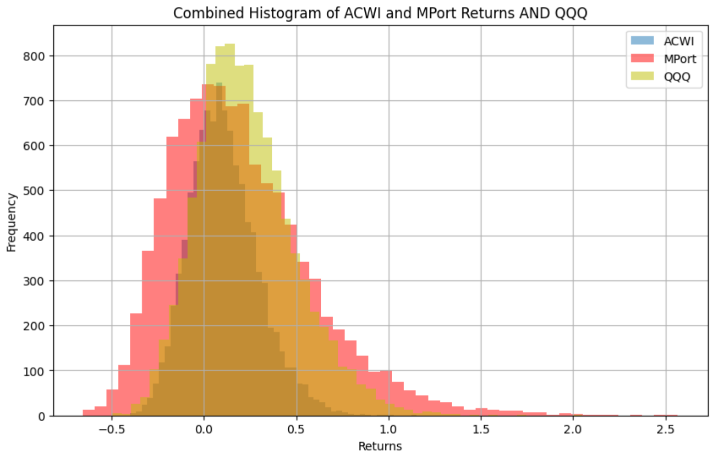
For the QQQ investment :

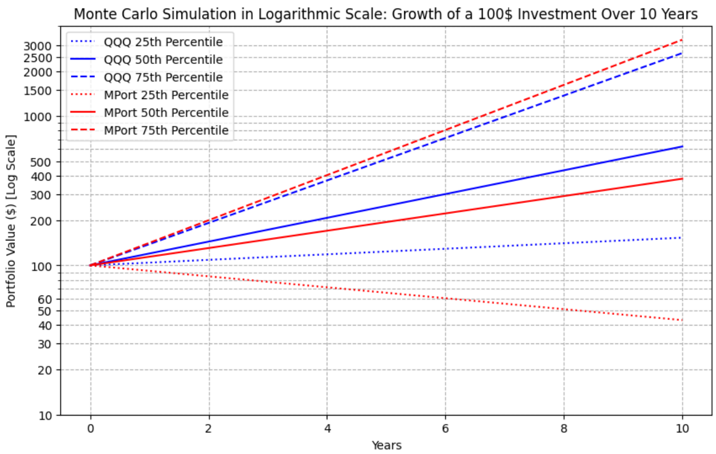
In the bullish case (50% probability), the average of the Top 75% and Record High is (38.75 + 205.10) / 2 = 121.925.
In the bearish case (50% probability), the average of the Bottom 25% and Record Low is (4.36 - 49.65) / 2 = -22.645.
The expected payoff for **QQQ** is then the average of the bullish and bearish cases: (121.925 + (-22.645)) / 2 ≈ **49.64**.

For the Benchmark ACWI :

In the bullish case (50% probability), the average of the Top 75% and Record High is (22.16 + 115.41) / 2 = 68.785.
In the bearish case (50% probability), the average of the Bottom 25% and Record Low is (-2.24 - 39.91) / 2 = -21.075.
The expected payoff for **ACWI** is then the average of the bullish and bearish cases: (68.785 + (-21.075)) / 2 ≈ **23.855**.

Since the expected payoff for **MPort** (**56.18**) is greater than the expected payoff for QQQ (**49.64**), **MPort** is the best strategy. Additionally, because these are the only two strategies available, and **MPort** yields the highest expected payoff, this constitutes a Bayesian Nash Equilibrium. In our strategy, we consider the market as an actor that sets the rules; we cannot predict its plans. We are driven by it since our strategy is correlated with the market. We do not sell short; we only invest by buying. However, we can still see that **QQQ** wins even in simulations of the lowest 25%. And according to our game, we would need to invest according to our method in the 10 stocks. But we have intentionally taken the records, in our game, to understand risk management properly. If we stick to our Monte Carlo simulation and only consider the distribution of the median, the bottom 25%, and the top 75%, here is what we obtain.





First, the histograms, reveal that our **MPort** stock portfolio is much more volatile, while the **QQQ** is much less so. Thanks to our analysis, we have already seen that both our choices outperform the market. The Monte Carlo simulation on the right shows an investment of 100$ over 10 years, and we can see that the estimated median returns are higher for the **QQQ** as well as the estimated returns from the bottom 25%. Only the estimated returns from the top 75% are higher, and the gap between the two is quite narrow. We also notice that the strategy of our portfolio is much riskier.

Conclusion and Limitations

Our game has shown us that game theory and statistics are complementary; the Bayesian game helped us make a decision. However, we can also see that from different perspectives, the decision is different. Our game has its weaknesses since it accounts for extreme values, but it was still very useful because it allowed us to apply our estimations and, above all, to model a response for ourselves. The model of our game is solid, but I think it could be improved with different market actions and perhaps focus on other quartiles. Nevertheless, from a general point of view, my choice between **QQQ** or my stocks would depend on other factors, for example, I would consider the cost of the ETF, the expense ratio, which are the management fees that go to the ETF's asset manager, in our case, Invesco. But also the fees from my broker, because if I have only 100$, maybe buying 10 shares and thus 10$ of each share would not be profitable if the fees are degressive or even if there is a minimum investment requirement.
Finally, it should be noted that **"Past performance does not guarantee future results."** And in reality, one should use other tools like Monte Carlo simulations. And given that even professionals make mistakes and no one can predict future movements of the financial markets, the answer will be given in months or years, to eventually see what the alpha of our portfolio **MPort** or **QQQ** was. Therefore, We'll have to keep a close eye on their prices to see how they will perform.

Appendix

A / Complete python code for our algorithm, detailed part by part.

1. Importing Data and Calculating Metrics
2. Performing Monte Carlo Simulation
3. Compiling Key Information
4. Detailing Monte Carlo Simulation
5. Visualising Returns Distribution
6. Advanced Monte Carlo Simulation Analysis
7. Conclusion and Future Work

B/ Reference list
C/ Data used

A / Complete python code for our algorithm, detailed part by part.
https://colab.research.google.com/drive/16mo2h4EEAy-wDNufM2eOn7mMwffnb4LZ?usp=sharing

1. First off, here we import official data through Yahoo Finance's API, including the stocks we want to analyze as well as QQQ and our benchmark, ACWI. We calculate the average annual return over 10 years and the volatility, which is the standard deviation.

```
import yfinance as yf
import numpy as np
import pandas as pd

tickers = ['ACWI', 'XOM', 'RYAAY', 'SU', 'CSCO', 'EBAY', 'UMC', 'PM', 'TRI', 'ACLS', 'MMT', 'QQQ']

start_date = '2014-03-01'
end_date = '2024-03-01'

data = yf.download(tickers, start=start_date, end=end_date)['Adj Close']

daily_returns = data.pct_change()

# Calculate average annual return
average_annual_return = ((1 + daily_returns.mean()) ** 252) - 1  # Compounded annual return

# Calculate annual volatility
annual_volatility = daily_returns.std() * np.sqrt(252)  # Standard deviation of daily returns scaled to annual

print("Average Annual Return:\n", average_annual_return)
print("\nAnnual Volatility:\n", annual_volatility)
```

```
[*********************100%%**********************]  12 of 12 completed
Average Annual Return:
 Ticker
ACLS     0.474149
ACWI     0.102651
CSCO     0.154202
EBAY     0.125370
MMT      0.064299
PM       0.091767
QQQ      0.209964
RYAAY    0.166274
SU       0.115927
TRI      0.223743
UMC      0.280587
XOM      0.094723
dtype: float64

Annual Volatility:
 Ticker
```

ACLS   0.513436
ACWI   0.169473
CSCO   0.249402
EBAY   0.297115
MMT   0.143496
PM   0.225441
QQQ   0.215269
RYAAY   0.345875
SU   0.378545
TRI   0.198466
UMC   0.370027
XOM   0.276342
dtype: float64

2. Then, we perform our Monte Carlo simulation. Please note that here, MPort is given from a separately calculated average using the above results. We conducted the simulation for all the stocks but actually only needed MPort, QQ, and ACWI. We could have simulated for all the stocks but chose 10, and it would have been too complicated to calculate and especially to visualise. Given that our portfolio of 10 stocks is evenly weighted, it wasn't worth it. So, here we simulated our portfolio based on the average returns and volatility.

```
import numpy as np
import pandas as pd

# Number of trading days per year in our analysis
trading_days = 252

Yahoo_average_returns = {
    'ACLS': 0.474149,
    'ACWI': 0.102651,
    'CSCO': 0.154202,
    'EBAY': 0.125370,
    'MMT': 0.064299,
    'PM': 0.091768,
    'RYAAY': 0.166274,
    'SU': 0.115927,
    'TRI': 0.223743,
    'UMC': 0.280587,
    'XOM': 0.094723,
    'MPort': 0.179104, #calculated separately, it's just the average of our 10 picks
    'QQQ': 0.209964,
}


Yahoo_volatilities = {
    'ACLS': 0.513436,
    'ACWI': 0.169473,
    'CSCO': 0.249402,
    'EBAY': 0.297115,
    'MMT': 0.143496,
    'PM': 0.225441,
    'RYAAY': 0.345875,
    'SU': 0.378545,
    'TRI': 0.198466,
    'UMC': 0.370027,
    'XOM': 0.276342,
    'MPort': 0.317724, #calculated separately, it's just the average of our 10 picks
    'QQQ': 0.215269,
}

# Monte Carlo Simulation
np.random.seed(42)  # For reproducible results
n_simulations = 10000  # Number of simulations to run
payoffs = pd.DataFrame()
# Simulate daily returns and calculate annual payoffs
```

```
for stock, average_return in Yahoo_average_returns.items():
    volatility = Yahoo_volatilities[stock]
    simulated_returns = np.random.normal(average_return / trading_days,
                                         volatility / np.sqrt(trading_days),
                                         (n_simulations, trading_days))
    final_value = (1 + simulated_returns).prod(axis=1)
    payoffs[stock] = final_value - 1

print(payoffs.describe())
```

```
            ACLS          ACWI          CSCO          EBAY           MMT  \
count  10000.000000  10000.000000  10000.000000  10000.000000  10000.000000
mean       0.595214      0.108341      0.167226      0.132711      0.064665
std        0.867114      0.187154      0.295733      0.342959      0.153468
min       -0.806112     -0.399066     -0.673063     -0.633428     -0.356257
25%        0.002147     -0.022395     -0.044940     -0.108696     -0.044371
50%        0.403264      0.092426      0.136024      0.080730      0.055230
75%        0.964279      0.221592      0.335011      0.319277      0.160758
max       14.183748      1.154065      2.342422      2.327340      0.861664

              PM         RYAAY            SU           TRI           UMC  \
count  10000.000000  10000.000000  10000.000000  10000.000000  10000.000000
mean       0.099111      0.185374      0.122822      0.248575      0.317713
std        0.251499      0.420429      0.437286      0.250053      0.501316
min       -0.519819     -0.676889     -0.768723     -0.448738     -0.743341
25%       -0.079778     -0.111982     -0.189134      0.073288     -0.041093
50%        0.070304      0.116394      0.045016      0.222546      0.236646
75%        0.247364      0.415385      0.349858      0.400694      0.578824
max        1.790190      3.115352      3.405364      1.562876      4.571982

             XOM         MPort           QQQ
count  10000.000000  10000.000000  10000.000000
mean       0.095934      0.199636      0.230476
std        0.305355      0.389091      0.266552
min       -0.607466     -0.656398     -0.496544
25%       -0.122017     -0.080956      0.043666
50%        0.053483      0.143015      0.201301
75%        0.269941      0.416415      0.387547
max        1.791319      2.568151      2.051022
```

3/ Next, we compiled the important information.

```
stats_ACWI = payoffs['ACWI'].describe()
print("Statistics for ACWI:")
print(stats_ACWI)
print("\n")

stats_QQQ = payoffs['QQQ'].describe()
print("Statistics for QQQ:")
print(stats_QQQ)
print("\n")

stats_MPort = payoffs['MPort'].describe()
print("Statistics for MPort (Portfolio):")
print(stats_MPort)
print("\n")
```

```
Statistics for ACWI:
count    10000.000000
mean         0.108341
std          0.187154
min         -0.399066
```

25%   -0.022395
50%   0.092426
75%   0.221592
max   1.154065
Name: ACWI, dtype: float64

Statistics for QQQ:
count   10000.000000
mean   0.230476
std   0.266552
min   -0.496544
25%   0.043666
50%   0.201301
75%   0.387547
max   2.051022
Name: QQQ, dtype: float64

Statistics for MPort (Portfolio):
count   10000.000000
mean   0.199636
std   0.389091
min   -0.656398
25%   -0.080956
50%   0.143015
75%   0.416415
max   2.568151
Name: MPort, dtype: float64

4/ Then, we detailed the Monte Carlo simulation as much as possible by offering different decompositions, which was useful for providing the graphical representation in the following part.

```
import pandas as pd
import numpy as np


percentiles_list = [i/100 for i in range(1, 100)]

# Statistics for ACWI
stats_ACWI = payoffs['ACWI'].describe(percentiles=percentiles_list)
print("Statistics for ACWI with Percentiles from 1% to 99%:")
print(stats_ACWI)
print("\n")

# Statistics for MPort
stats_MPort = payoffs['MPort'].describe(percentiles=percentiles_list)
print("Statistics for MPort (Portfolio) with Percentiles from 1% to 99%:")
print(stats_MPort)
print("\n")

# Statistics for QQQ
stats_QQQ = payoffs['QQQ'].describe(percentiles=percentiles_list)
print("Statistics for QQQ with Percentiles from 1% to 99%:")
print(stats_QQQ)
print("\n")
```

Statistics for ACWI with Percentiles from 1% to 99%:

```
count    10000.000000
mean         0.108341
std          0.187154
min         -0.399066
1%          -0.263029
             ...
96%          0.475057
97%          0.505089
98%          0.547895
99%          0.617760
max          1.154065
Name: ACWI, Length: 104, dtype: float64
```

Statistics for MPort (Portfolio) with Percentiles from 1% to 99%:

```
count    10000.000000
mean         0.199636
std          0.389091
min         -0.656398
1%          -0.456832
             ...
96%          0.992319
97%          1.062678
98%          1.177644
99%          1.380633
max          2.568151
Name: MPort, Length: 104, dtype: float64
```

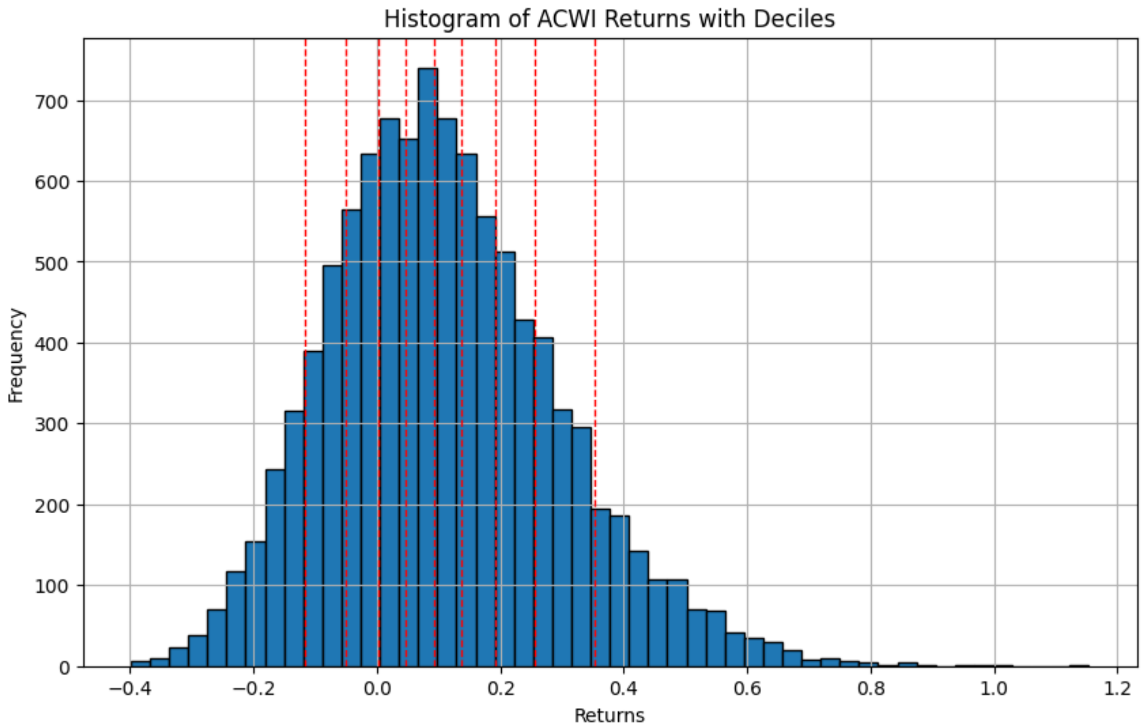
Statistics for QQQ with Percentiles from 1% to 99%:

```
count    10000.000000
mean         0.230476
std          0.266552
min         -0.496544
1%          -0.275135
             ...
96%          0.755772
97%          0.804687
98%          0.871907
99%          0.968015
max          2.051022
Name: QQQ, Length: 104, dtype: float64
```

5/ Indeed, here we visualise our histogram to see the frequency of returns. We separated it into deciles but could have used other distributions. We didn't directly use them in our report, but these representations were very helpful for its elaboration.

```
import matplotlib.pyplot as plt
import numpy as np

acwi_data = payoffs['ACWI']

deciles = np.percentile(acwi_data, [10, 20, 30, 40, 50, 60, 70, 80, 90])

plt.figure(figsize=(10, 6))
plt.hist(acwi_data, bins=50, alpha=1, edgecolor='black')

for decile in deciles:
    plt.axvline(x=decile, color='red', linestyle='dashed', linewidth=1)

plt.title('Histogram of ACWI Returns with Deciles')
plt.xlabel('Returns')
plt.ylabel('Frequency')
```

```
plt.grid(True)
plt.show()

import numpy as np

acwi_data = payoffs['ACWI']

deciles = np.percentile(acwi_data, np.arange(10, 100, 10))

for i, decile in enumerate(deciles, 1):
    print(f"Decile {i}: {decile}")
```

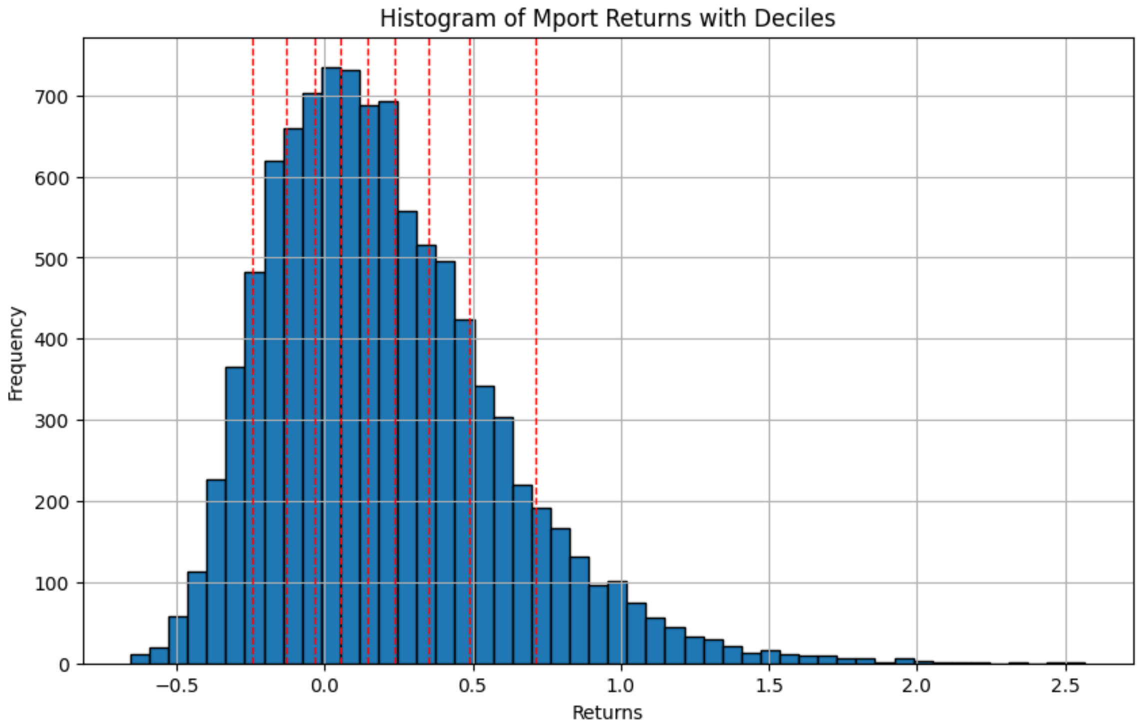


Decile 1: -0.117651514024291 96
Decile 2: -0.0494248198470473 85
Decile 3: 0.0020699451433774368
Decile 4: 0.04743920376121396
Decile 5: 0.09242582044705305
Decile 6: 0.13780647337655472
Decile 7: 0.19132426656015245
Decile 8: 0.2567403007474688
Decile 9: 0.3529060793629938

We do the same as for the ACWI but this time for MPort, our 10 stocks.

```
import matplotlib.pyplot as plt
import numpy as np

MPort_data = payoffs['MPort']

deciles = np.percentile(MPort_data, [10, 20, 30, 40, 50, 60, 70, 80, 90])

plt.figure(figsize=(10, 6))
plt.hist(MPort_data, bins=50, alpha=1, edgecolor='black')

for decile in deciles:
    plt.axvline(x=decile, color='red', linestyle='dashed', linewidth=1)

plt.title('Histogram of Mport Returns with Deciles')
```

```
plt.xlabel('Returns')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()

import numpy as np

MPort_data = payoffs['MPort']

deciles = np.percentile(MPort_data, np.arange(10, 100, 10))

for i, decile in enumerate(deciles, 1):
    print(f"Decile {i}: {decile}")
```

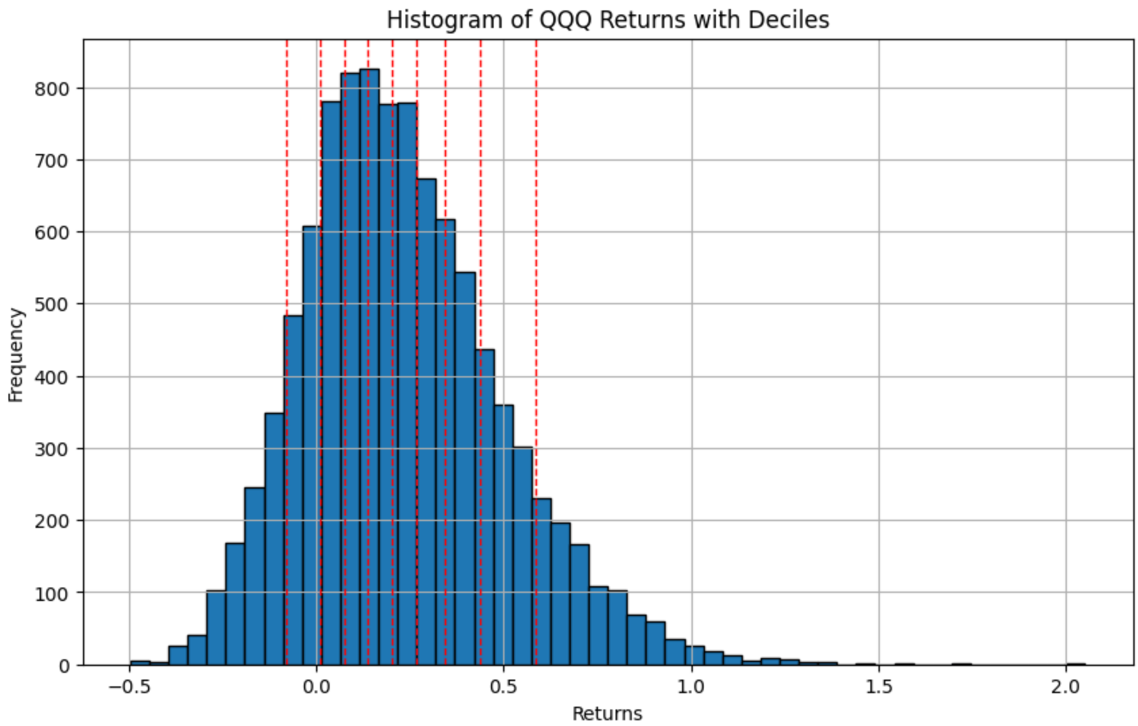


Decile 1: -0.24221129387068532
Decile 2: -0.1292200000216042
Decile 3: -0.03511071451322695
Decile 4: 0.05333051544997696
Decile 5: 0.14301470494744783
Decile 6: 0.23484107654749561
Decile 7: 0.3501479688491674
Decile 8: 0.4890494552614071
Decile 9: 0.7108485830646794

Then we do the same for QQQ, which is useful for clearly visualising the distribution. Adding the red lines and representing the deciles helps to visualise the differences between the assets well.

```
import matplotlib.pyplot as plt
import numpy as np

acwi_data = payoffs['QQQ']

deciles = np.percentile(acwi_data, [10, 20, 30, 40, 50, 60, 70, 80, 90])

plt.figure(figsize=(10, 6))
plt.hist(acwi_data, bins=50, alpha=1, edgecolor='black')

for decile in deciles:
    plt.axvline(x=decile, color='red', linestyle='dashed', linewidth=1)
```

```
plt.title('Histogram of QQQ Returns with Deciles')
plt.xlabel('Returns')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()

import numpy as np

acwi_data = payoffs['QQQ']

deciles = np.percentile(acwi_data, np.arange(10, 100, 10))

for i, decile in enumerate(deciles, 1):
    print(f"Decile {i}: {decile}”)
```

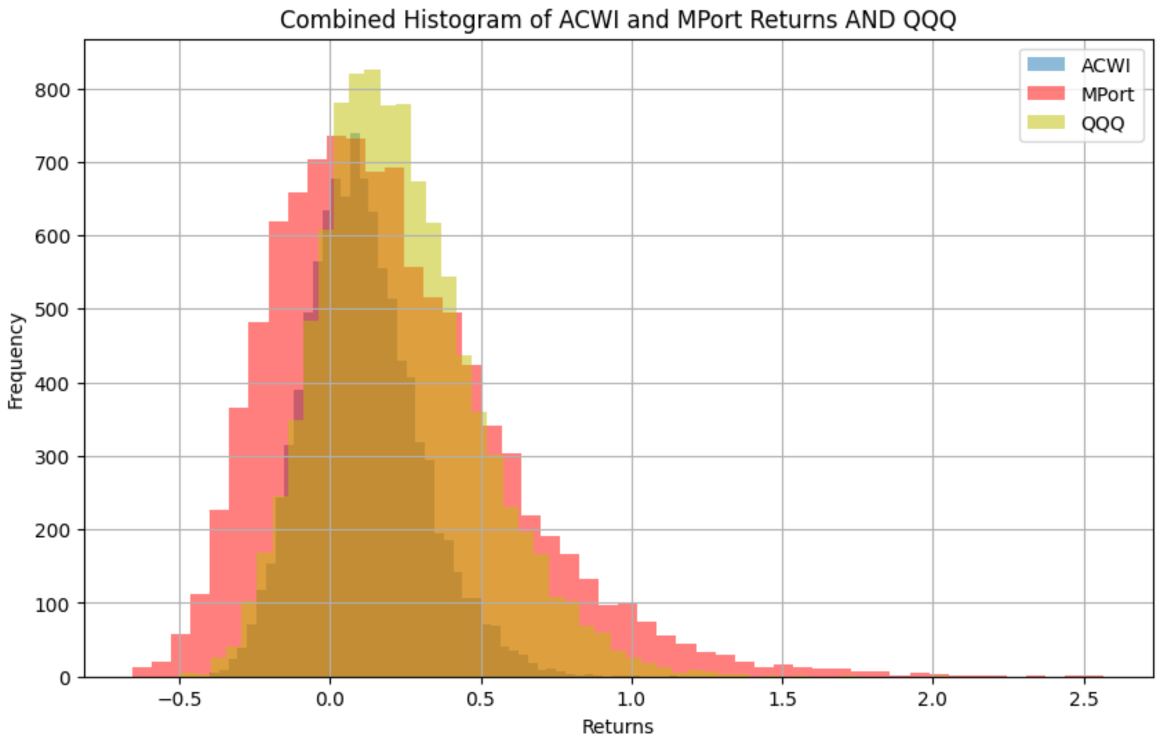


Decile 1: -0.081914384441351857
Decile 2: 0.009979313083195801
Decile 3: 0.0762652919828918
Decile 4: 0.1361237845023306
Decile 5: 0.2013009833968763
Decile 6: 0.2670141827187233
Decile 7: 0.34348018900060806
Decile 8: 0.43746299391547233
Decile 9: 0.5863613375643573

Finally, here we group the three histograms to compare the assets well, and this chart was used in our report.

```
import matplotlib.pyplot as plt
import seaborn as sns

acwi_data = payoffs['ACWI']
mport_data = payoffs['MPort']
QQQ_data = payoffs['QQQ']

plt.figure(figsize=(10, 6))
plt.hist(acwi_data, bins=50, alpha=0.5, label='ACWI')
```

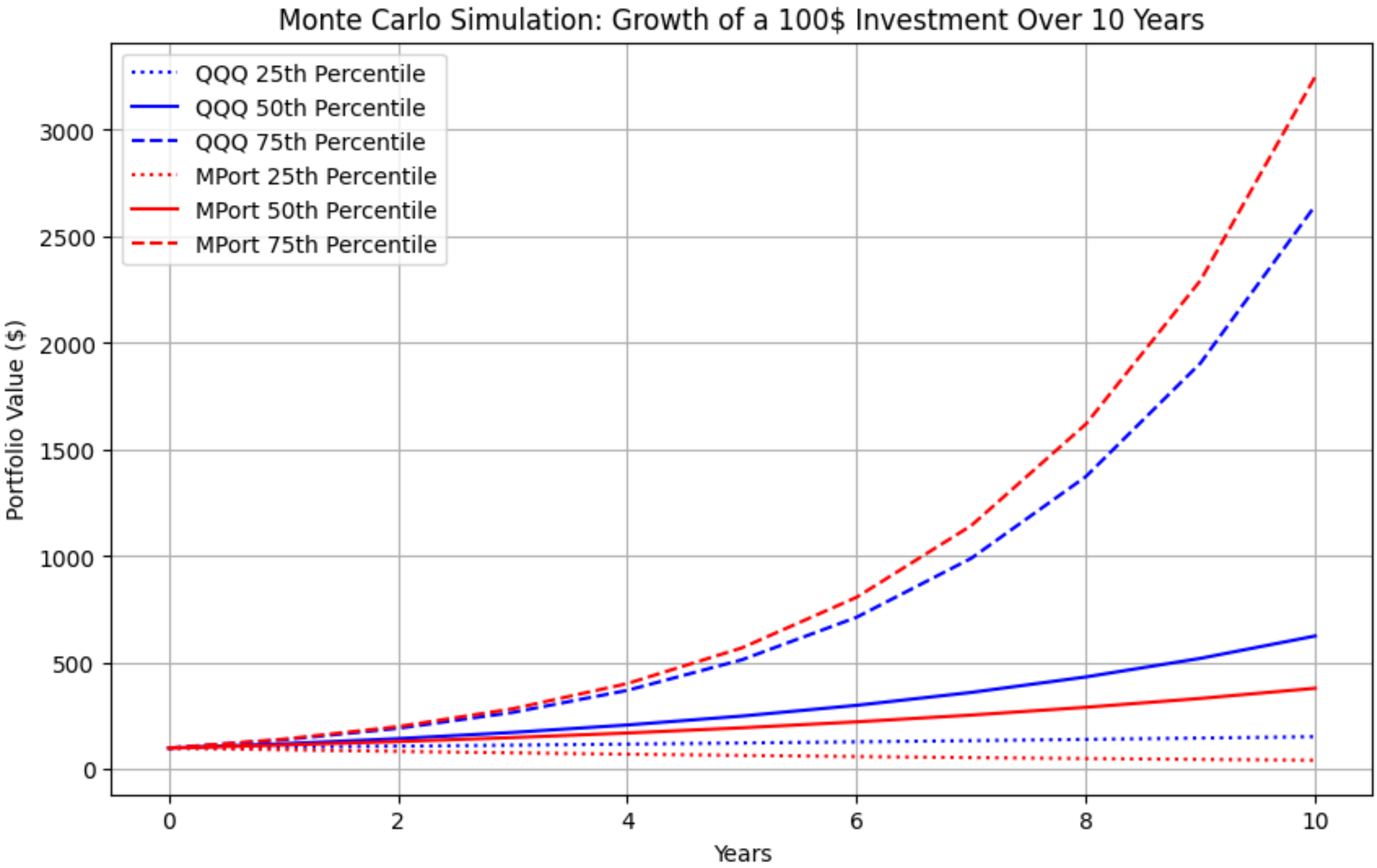
```
plt.hist(QQQ_data, bins=50, alpha=0.5, label='QQQ', color='y')
plt.title('Combined Histogram of ACWI and MPort Returns AND QQQ')
plt.xlabel('Returns')
plt.ylabel('Frequency')
plt.legend()
plt.grid(True)
plt.show()
```

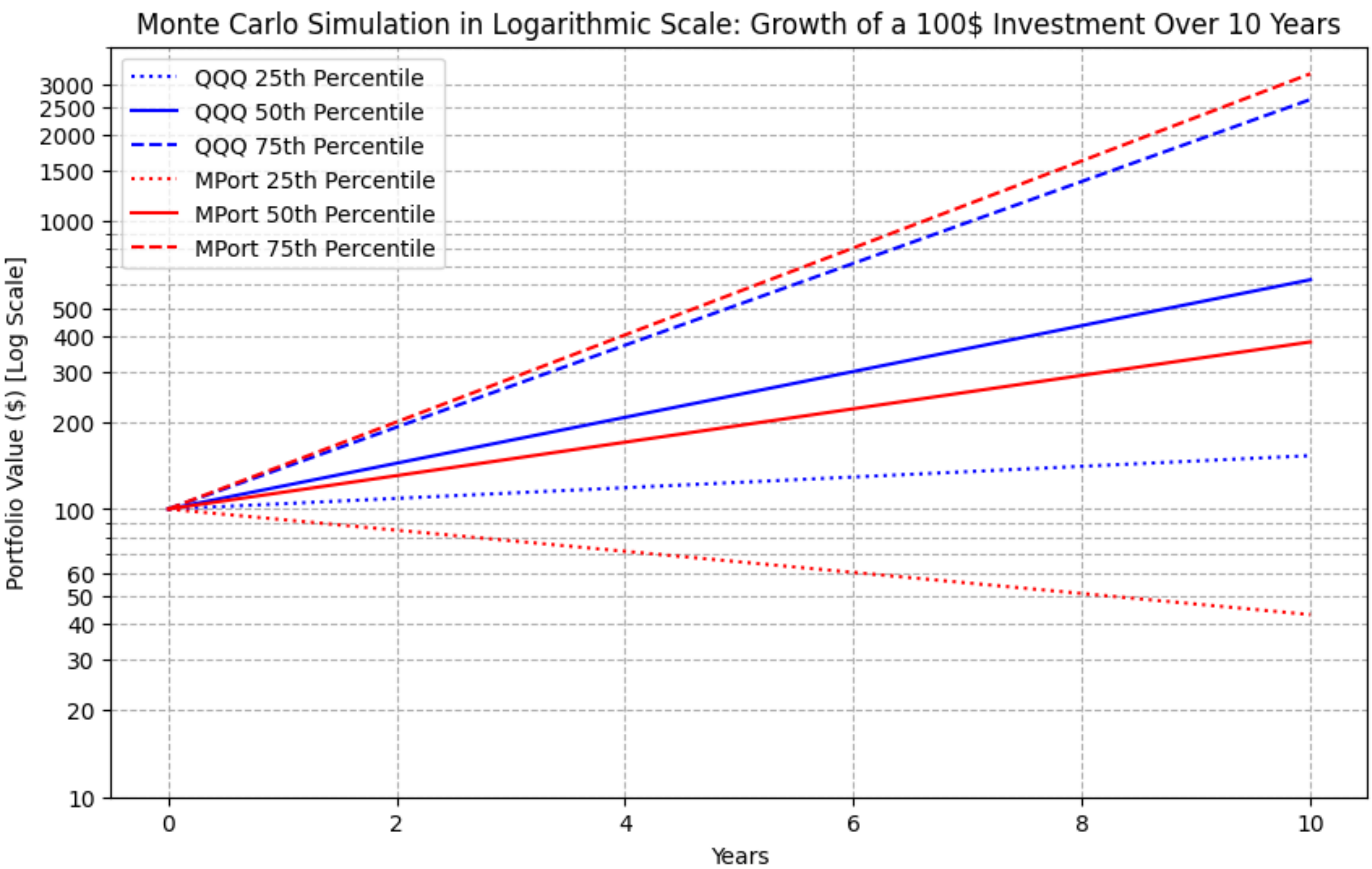


6/ Lastly, here we represented our Monte Carlo simulation for QQQ and MPort, taking the top 75% and bottom 75%. Of course, we could take other deciles or distributions.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter

# Provided data for ACWI, QQQ, MPort for the 25th, 50th, and 75th percentiles
percentiles_qqq = [0.043666, 0.201301, 0.387547]
percentiles_mport = [-0.080956, 0.143015, 0.416415]

# Simulation period set for 10 years
years = np.arange(0, 11, 1)

# Monte Carlo simulation function for initial investment over 10 years
def simulate_growth(initial_investment, percentile, years):
    """
    Calculates the growth of the initial investment for given percentiles over the period of years.

    :param initial_investment: The initial investment
    :param percentile: The return percentile (25%, 50%, 75%)
    :param years: The duration of the period in years
    :return: An array with the investment value for each year
    """
    return np.array([initial_investment * ((1 + percentile) ** year) for year in years])


growth_qqq_25 = simulate_growth(100, percentiles_qqq[0], years)
```

```
growth_qqq_75 = simulate_growth(100, percentiles_qqq[2], years)

growth_mport_25 = simulate_growth(100, percentiles_mport[0], years)
growth_mport_50 = simulate_growth(100, percentiles_mport[1], years)
growth_mport_75 = simulate_growth(100, percentiles_mport[2], years)

plt.figure(figsize=(10, 6))

plt.plot(years, growth_qqq_25, label='QQQ 25th Percentile', color='blue', linestyle='dotted')
plt.plot(years, growth_qqq_50, label='QQQ 50th Percentile', color='blue', linestyle='solid')
plt.plot(years, growth_qqq_75, label='QQQ 75th Percentile', color='blue', linestyle='dashed')

plt.plot(years, growth_mport_25, label='MPort 25th Percentile', color='red', linestyle='dotted')
plt.plot(years, growth_mport_50, label='MPort 50th Percentile', color='red', linestyle='solid')
plt.plot(years, growth_mport_75, label='MPort 75th Percentile', color='red', linestyle='dashed')

plt.title('Monte Carlo Simulation: Growth of a 100$ Investment Over 10 Years')
plt.xlabel('Years')
plt.ylabel('Portfolio Value ($)')
plt.legend()
plt.grid(True)
plt.show()

# Function to format the y-axis labels as standard numbers
def to_standard_format(x, pos):
    return '{:0.0f}'.format(x)


# Creating the chart in logarithmic scale for easier representation
plt.figure(figsize=(10, 6))
formatter = FuncFormatter(to_standard_format)

plt.semilogy(years, growth_qqq_25, label='QQQ 25th Percentile', color='blue', linestyle='dotted')
plt.semilogy(years, growth_qqq_50, label='QQQ 50th Percentile', color='blue', linestyle='solid')
plt.semilogy(years, growth_qqq_75, label='QQQ 75th Percentile', color='blue', linestyle='dashed')

plt.semilogy(years, growth_mport_25, label='MPort 25th Percentile', color='red', linestyle='dotted')
plt.semilogy(years, growth_mport_50, label='MPort 50th Percentile', color='red', linestyle='solid')
plt.semilogy(years, growth_mport_75, label='MPort 75th Percentile', color='red', linestyle='dashed')

plt.gca().yaxis.set_major_formatter(formatter) # here we can easily modify the values of the ordinate axis.
plt.gca().set_yticks([10, 20, 30, 40, 50, 60, 100, 200, 300, 400, 500, 1000, 1500, 2000, 2500, 3000])


plt.title('Monte Carlo Simulation in Logarithmic Scale: Growth of a 100$ Investment Over 10 Years')
plt.xlabel('Years')
plt.ylabel('Portfolio Value ($) [Log Scale]')
plt.legend()
plt.grid(True, which="both", ls="--")
plt.show()
```


Monte Carlo Simulation: Growth of a 100$ Investment Over 10 Years
QQQ 25th Percentile
QQQ 50th Percentile
QQQ 75th Percentile
MPort 25th Percentile
MPort 50th Percentile
MPort 75th Percentile
Portfolio Value ($)
0
500
1000
1500
2000
2500
3000
0
2
4
6
8
10
Years


Monte Carlo Simulation in Logarithmic Scale: Growth of a 100$ Investment Over 10 Years
QQQ 25th Percentile
QQQ 50th Percentile
QQQ 75th Percentile
MPort 25th Percentile
MPort 50th Percentile
MPort 75th Percentile
Portfolio Value ($) [Log Scale]
10
20
30
40
50
60
100
200
300
400
500
1000
1500
2000
2500
3000
0
2
4
6
8
10
Years

7/ To conclude, our program allows an overview of the selected assets in the Monte Carlo simulation. Moving forward, we could create a script to automate our Bayesian game provided in our report to automate the game and analyse several stocks and indices according to many deciles or quartiles, and automate our analyses subsequently. The project is scalable, and I sincerely plan to use it in the future to automate and improve it by adding other functionalities.

B/ Reference list

Blackrock (2023). Market Neutral Investing. [online] BlackRock. Available at: https://www.blackrock.com/us/individual/insights/market-neutral-investing

Chen, J. (2023). Alpha: What It Means in Investing, With Examples. [online] Investopedia. Available at: https://www.investopedia.com/terms/a/alpha.asp

Guzun, E. (2023). Unraveling the Truth About Market-Neutral Equity Strategies. [online] HedgeNordic. Available at: https://hedgenordic.com/2023/06/unraveling-the-truth-about-market-neutral-equity-strategies/

Invesco (2023). Invesco QQQ ETF Performance. [online] www.invesco.com. Available at: https://www.invesco.com/qqq-etf/en/performance.html

IShares (2024). iShares MSCI ACWI UCITS ETF. [online] BlackRock. Available at: https://www.blackrock.com/fr/intermediaries/products/251850/ishares-msci-acwi-ucits-etf

Khanna, S. (2023). 68% equity schemes underperform their benchmarks in five years. The Economic Times. [online] 4 Dec. Available at: https://economictimes.indiatimes.com/mf/analysis/68-equity-schemes-underperform-their-benchmarks-in-five-years/articleshow/105717226.cms?from=mdr

Officialdata.org (2023). S&P 500 Returns since 2003. [online] www.officialdata.org. Available at: https://www.officialdata.org/us/stocks/s-p-500/2003?amount=100&endYear=2023

C/ Data used

https://finance.yahoo.com/quote/XOM/

https://finance.yahoo.com/quote/SU/

https://finance.yahoo.com/quote/EBAY/

https://finance.yahoo.com/quote/PM/

https://finance.yahoo.com/quote/ACLS/

https://finance.yahoo.com/quote/RYAAY/

https://finance.yahoo.com/quote/CSCO/

https://finance.yahoo.com/quote/UMC/

https://finance.yahoo.com/quote/TRI/

https://finance.yahoo.com/quote/MMT.PA/