The Risky Business of Safe Investing: Optimizing Conservative Portfolios

Ian Lucas and Carrie Little

Shiley-Marcos School of Engineering, University of San Diego

AAI500: Probability and Statistics for Artificial Intelligence

Leonid Shpaner, M.S.

AAI-500-02: Final Project Group 9

October 21, 2024

The Risky Business of Safe Investing: Optimizing Conservative Portfolios

Introduction

Due to a stagnant economy coupled with rising interest rates in the year 2022, bonds and stocks experienced a dramatic increase in their correlation while falling in value simultaneously. This led to a breakdown in diversification and significant losses in ostensibly conservative portfolios. Figure 1 shows this for the supposedly safe Vanguard LifeStrategy Income Fund (VASIX), which consists of 80% global bonds and 20% global stocks. VASIX experienced a 17% drawdown and still has not recovered its value from three years ago. Figure 2 depicts the correlation between stocks and bonds, as represented by the rolling 12-month correlation between the Vanguard Total World Stock ETF (VT) and PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ). The correlation started 2022 near zero, but rose above 0.50 over the course of the year and has remained high ever since. While the nearly 10-year correlation between the two is a low 0.11, the instability in this relationship revealed a vulnerability that became manifest in 2022.

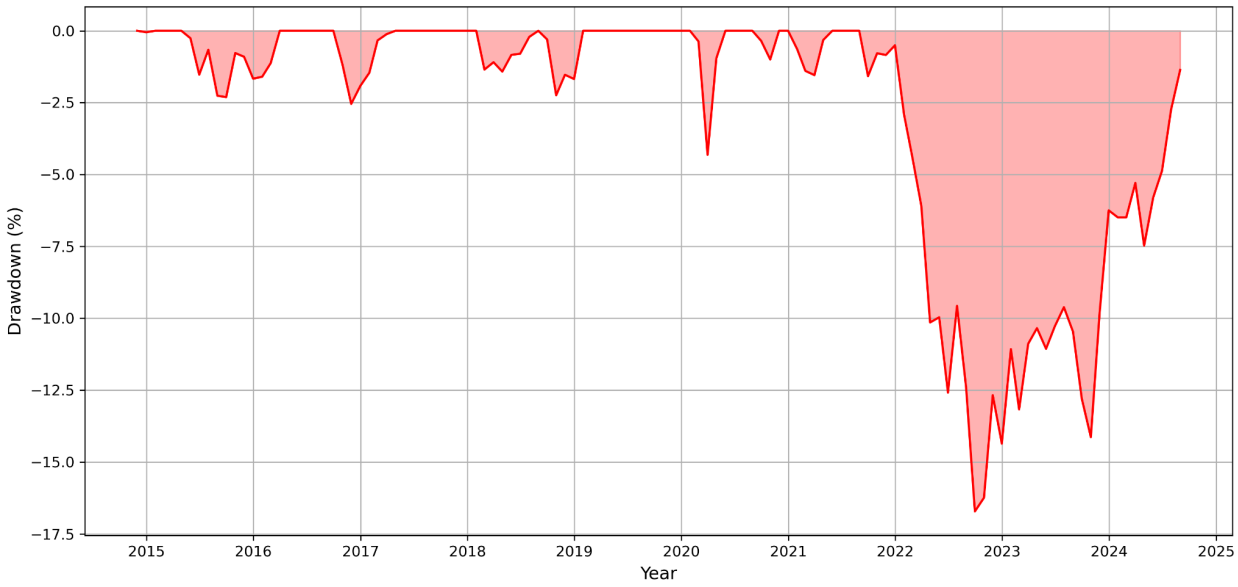


Figure 1 - Drawdown Analysis for VASIX

Figure 2 - Rolling 12-month Correlation Between Stocks and Bonds

In response to significant losses experienced by our clients in 2022 within their conservative asset allocation strategies, our asset management firm seeks to explore more robust portfolio strategies. Many clients shifted their investments to Treasury Bills when interest rates surpassed 4.0%, finding solace in these seemingly safer options. However, with the Federal Reserve beginning to lower interest rates, clients are looking for new strategies that offer total returns of 4.0% or more without the substantial drawdowns they encountered in 2022.

In this report, we investigate a variety of liquid investments across a spectrum of correlation profiles. By combining these diverse assets, we can meet the needs of our clients, offering better risk-adjusted returns. We employ portfolio optimization techniques to evaluate Risk Parity, Minimum Variance, and Maximum Diversification approaches. Our analysis concludes that a Risk Parity allocation offers the most balanced risk-return profile, avoiding the exclusion of any particular asset class, and better aligning with the evolving needs of our clients.

Data Cleaning/Preparation

For this analysis, we gathered monthly total return data for 15 mutual funds and exchange-traded funds (ETFs) from Portfolio Visualizer. The dataset covers the period from November 2014 through August 2024–the longest common time period in which all the funds were available. There are 118 months of data for each asset. Vanguard LifeStrategy Income Fund (VASIX) serves as the benchmark for this study and the other 14 assets represent our opportunity set.

The dataset was formatted with monthly returns in decimal form, with no missing data points. Because the data came from a single source in a standardized format, minimal preparation was required. However, in a different research context, we might have needed to handle missing data points, different data types, multiple data formats, currency conversions, and inconsistent timeframes. Additionally, we might have considered data smoothing techniques to account for outliers, particularly in highly volatile assets like Bitcoin. For our present analysis, we used the raw data to reflect real-world outcomes for publicly available investment products, but later will discuss extending the analysis with additional data of varying types. Future analysis may also warrant data transformations in order to satisfy the assumptions of various statistical models.

```
DatetimeIndex: 118 entries, 2014-11-30 to 2024-08-31
Data columns (total 15 columns):
 #   Column                                          Non-Null Count  Dtype
---  ------                                          --------------  -----
 0   Vanguard LifeStrategy Income Fund (VASIX)       118 non-null    float64
 1   Vanguard Total World Stock ETF (VT)             118 non-null    float64
 2   PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)    118 non-null    float64
 3   AQR Diversified Arbitrage I (ADAIX)             118 non-null    float64
 4   iShares Gold Trust (IAU)                        118 non-null    float64
 5   Bitcoin Market Price USD (^BTC)                 118 non-null    float64
 6   AQR Risk-Balanced Commodities Strategy I (ARCIX)  118 non-null  float64
 7   AQR Long-Short Equity I (QLEIX)                 118 non-null    float64
 8   AQR Style Premia Alternative I (QSPIX)          118 non-null    float64
 9   AQR Equity Market Neutral I (QMNIX)             118 non-null    float64
 10  AQR Macro Opportunities I (QGMIX)               118 non-null    float64
 11  AGF U.S. Market Neutral Anti-Beta (BTAL)        118 non-null    float64
 12  AQR Managed Futures Strategy HV I (QMHIX)       118 non-null    float64
 13  Invesco DB US Dollar Bullish (UUP)              118 non-null    float64
 14  ProShares VIX Mid-Term Futures (VIXM)           118 non-null    float64
```

Table 1 - Dataset Info

Exploratory Data Analysis

Our exploratory data analysis evaluates each asset's potential role in diversification. Later, we will put this to the test with a variety of optimization methods that aim to reduce overall portfolio risk below that of the benchmark while preserving a satisfactory return profile through shifting economic environments. First, however, we need to deepen our understanding of the statistical properties of each asset.

Correlations

Because our focus is on diversification, we begin with an examination of the correlation matrix, which tells us a high level story about the relationships between all of the assets. We are most interested in understanding two things with regards to the correlations.

First, we want to know how correlated each asset is with respect to the benchmark, VASIX. To the extent that the assets provide a diversification benefit, they will need to have a relatively low correlation to VASIX. This is important because, while stocks and bonds have a near-zero correlation to one another over time, that correlation is not static, as we saw earlier in Figure 2. Our hope is that a number of the assets exhibit low and possibly even negative correlations with VASIX (and, by extension, with stocks and bonds). If they do, then it hints at the potential to mitigate extreme negative events like those experienced in 2022.

Second, we want to understand the cross-correlations among the assets. Ideally, they have low correlation with one another that is stable over time. This is critical for the reason that we examined at the start of this paper. Namely, although stocks and bonds exhibited a low overall correlation, their relationship was not stable, leading to a breakdown in diversification and steep losses.

To those ends, the correlation matrix in Figure 3 helps us visualize the potential diversity available to us in our opportunity set (note that we are only displaying the lower triangle of the correlation matrix). We make a number of initial observations.

- On average, the 14 assets have a 0.03 correlation with VASIX. This suggests that they may provide meaningful diversification from a conventional conservative allocation such as that provided by VASIX.
- VASIX and the Vanguard Total World Stock ETF (VT) have high positive correlation (0.80), which makes sense because the global stock allocation of VT is a part of VASIX.
- The PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ) also has a relatively high correlation with VASIX of 0.62, which also makes sense because the global bond component of VASIX is sensitive to changes in interest rates, to which ZROZ is especially sensitive.
- Promisingly, the next highest correlation with VASIX comes from AQR Diversified Arbitrage I (ADAIX) fund and iShares Gold Trust (IAU), coming in at a modest 0.39 and 0.37, respectively.
- From there, the correlations with VASIX drop off quickly, with seven assets demonstrating a negative correlation as low as -0.51 for the Invesco DB US Dollar Bullish (UUP) fund.

Here are some highlights regarding cross-correlations among the assets in the opportunity set (not including VASIX):

- The average cross-correlation of the 14 assets (excluding the correlation of each asset with itself) also is 0.03, indicating that they will work well with one another.
- Three of the 14 have a high correlation above 0.70 with one another–AQR Long-Short Equity I (QLEIX), AQR Equity Market Neutral I (QMNIX), and AQR Style Premia Alternative I (QSPIX).
- There are two other instances of cross-correlations slightly above 0.50–the AQR Managed Futures Strategy HV (QMHIX) fund has a 0.56 correlation with the AQR Global Macro Opportunities (QGMIX) while ADAIX has a 0.51 correlation with VT.

- Numerous assets have negative correlations with one another. For example, ProShares VIX Mid-Term Futures (VIXM), AGF. U.S. Market Neutral Anti-Beta (BTAL), and UUP all have correlations below -0.50 with global stocks (VT).

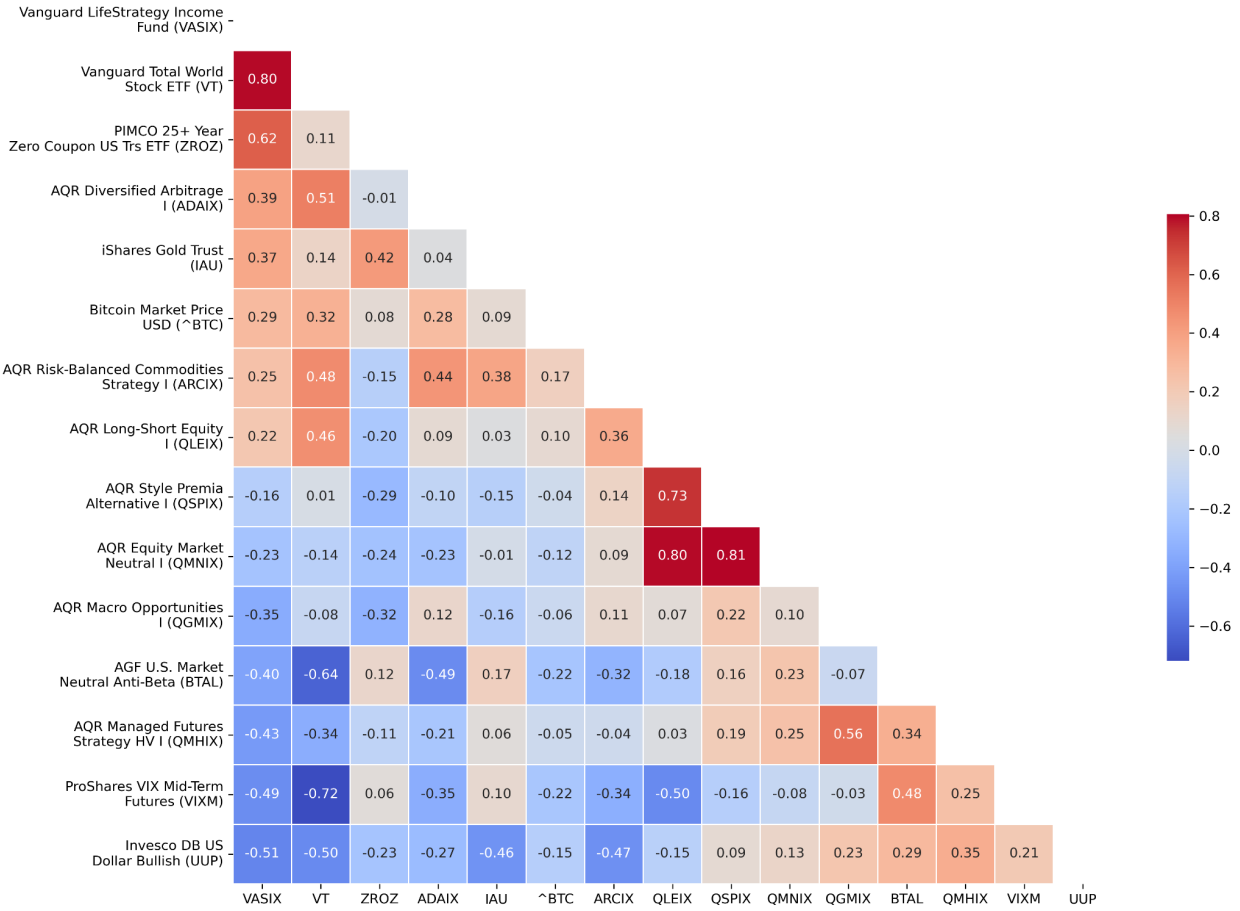


Figure 3 - Correlation Matrix Heatmap (Lower Triangle)

Looking under the hood, Figure 4, on the next page, shows that the average cross-correlation was low throughout the period studied. The highest it ever reached was a mere 0.12 in late 2017. In 2022, it oscillated around 0.00, indicating that diversification was maintained during this critical time frame.

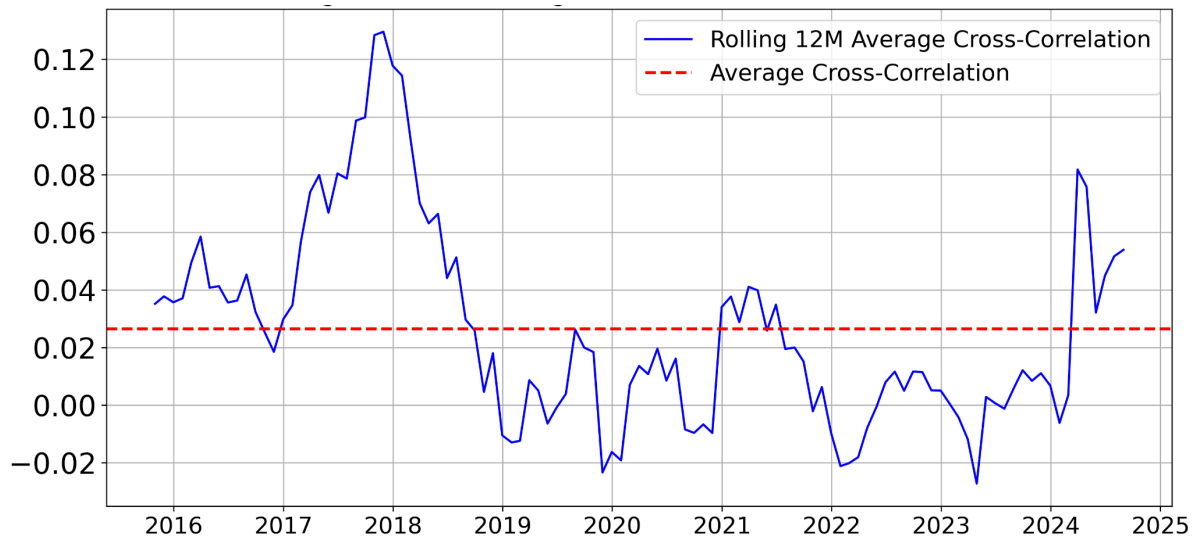


Figure 4 - Rolling 12-month Average Asset Cross-Correlation (excluding VASIX)

Monthly Risk and Return Summary

From correlations, our exploration moves on to consider the basic return characteristics in the assets. Turning our attention to Table 2, which is sorted in descending order of each asset's correlation with the benchmark, we make a number of initial observations related to risk and reward. We will start by considering the risk characteristics of the monthly returns as represented by standard deviation, minimum 1-month return, and skew. Then we will consider the reward side of the coin in terms of the arithmetic average (mean) monthly return.

VASIX has a lower monthly standard deviation (1.63%) than every other asset, indicating that it is a relatively conservative investment. Similarly, its lowest 1-month return (-4.92%), while steep, was less severe than 13 of the 14 assets–only UUP had a slightly less bad worst month (-4.73%). In fact, six of the assets had minimum 1-month returns below -10.00%. The higher volatility and worse minimum returns tells us that the assets are individually quite risky. The seemingly favorable correlations noted above will be put to the test if the goal is to achieve a better risk profile than VASIX.

Interestingly, the skew of VASIX was -0.24, which is worse than all but two of the other assets. This indicates the presence of sharp losses for VASIX relative to its full distribution of returns, which we already know were experienced in 2022 and will explore in more detail later in our analysis. The fact that 11 of the 14 other assets had positive skew provides some hope that, when combined, they may truncate the left tail and mitigate severe losses. But it is too early to get our hopes up because it is not yet clear if their steeper individual losses and higher volatility can be diversified away. That the maximum monthly return for 10 of the 14 assets exceeded 10.00%, coupled with the fact that the mean return was above the median for 11 of the 14 assets, is indicative of the positive skew exhibited by many of them.

The mean monthly return for 11 of the 14 assets is equal to or greater than VASIX's 0.29% monthly return (with BTC being notable for its extreme monthly return of 6.67%). Only one, VIXM, had a negative mean, suggesting that its strong diversification potential (stemming from its -0.49 correlation with VASIX) may come at the cost of a lower return. However, the fact that most assets have a higher mean return suggests the possibility that they could offer a reasonable return relative to VASIX while pursuing diversification.

One interesting observation is that the mean return tends to be quite small compared to the standard deviation. This is important because investors care about *compounded* (geometric) returns, which are always lower than arithmetic mean returns in the presence of positive variance. Consider an extreme example to illustrate the point. Suppose and investment rose 50%, increasing a $10,000 starting value to $15,000. Then, in the second period, it fell by 50%, cutting the value down to $7,500. The arithmetic mean (+50%, -50%) is zero, but the compounded growth rate is negative. Because VASIX has a low standard deviation, it suggests that it may suffer from less of a return drag due to volatility than the other assets. We will see later that this phenomenon results in VASIX having a larger

compounded return than six of the other assets owing to its low standard deviation, whereas in arithmetic mean terms, it only surpasses one other asset.

One quick note is that the positive kurtosis seen in most assets, coupled with their skew, is a sign that their returns are not normally distributed, which we will explore in more depth below.

Summary Statistics of Asset Returns (in Percent Form, Including Skewness and Kurtosis, Sorted by Correlation with VASIX)

	Count	Correlation with VASIX	Mean	Std Dev	Min	25%	Median	75%	Max	Skewness	Kurtosis
Vanguard LifeStrategy Income Fund (VASIX)	118	1.00	0.29	1.63	-4.92	-0.29	0.47	0.97	5.04	-0.24	1.59
Vanguard Total World Stock ETF (VT)	118	0.80	0.84	4.35	-14.76	-1.88	1.23	3.05	12.37	-0.43	1.02
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)	118	0.62	0.14	6.08	-13.43	-4.13	0.08	3.77	17.59	0.38	0.33
AQR Diversified Arbitrage I (ADAIX)	118	0.39	0.44	1.69	-8.15	-0.33	0.43	0.98	7.43	0.09	8.49
iShares Gold Trust (IAU)	118	0.37	0.70	4.04	-8.36	-1.97	-0.04	2.92	11.22	0.38	-0.07
Bitcoin Market Price USD (^BTC)	118	0.29	6.67	22.12	-40.60	-8.01	4.68	18.23	72.00	0.61	0.60
AQR Risk-Balanced Commodities Strategy I (ARCIX)	118	0.25	0.55	4.69	-14.42	-2.25	0.16	3.95	12.17	0.11	0.13
AQR Long-Short Equity I (QLEIX)	118	0.22	0.89	3.30	-8.21	-1.09	0.92	2.54	11.58	0.02	1.27
AQR Style Premia Alternative I (QSPIX)	118	-0.16	0.54	3.88	-7.81	-1.48	0.00	1.75	14.08	1.03	1.82
AQR Equity Market Neutral I (QMNIX)	118	-0.23	0.53	2.87	-6.14	-1.03	0.15	2.10	11.07	0.65	1.39
AQR Macro Opportunities I (QGMIX)	118	-0.35	0.29	2.18	-7.12	-0.85	0.21	1.51	6.68	0.10	1.27
AGF U.S. Market Neutral Anti-Beta (BTAL)	118	-0.40	0.19	4.29	-14.96	-2.18	-0.07	2.68	9.48	-0.37	1.01
AQR Managed Futures Strategy HV I (QMHIX)	118	-0.43	0.36	4.74	-8.85	-3.01	-0.10	3.60	12.75	0.30	-0.26
ProShares VIX Mid-Term Futures (VIXM)	118	-0.49	-0.85	9.36	-18.01	-5.92	-1.81	2.08	62.89	2.91	17.90
Invesco DB US Dollar Bullish (UUP)	118	-0.51	0.28	1.91	-4.73	-1.30	0.40	1.64	4.84	-0.08	-0.30

Table 2 - Asset Summary Statistics

Monthly Return Time Series Plots

Figure 5 plots the monthly return time series of each asset over the nearly 10-year period. The main takeaway from this is that returns vary wildly from month to month, which is in keeping with the standard deviations being so high relative to the mean, as we saw in Table 2. Sometimes assets can go years with relatively “quiet” returns that belie their embedded risks before sudden spikes in volatility bring those risks to the forefront–VASIX, ADAIX, and QSPIX are good examples of this, though all of them how this property to some extent.

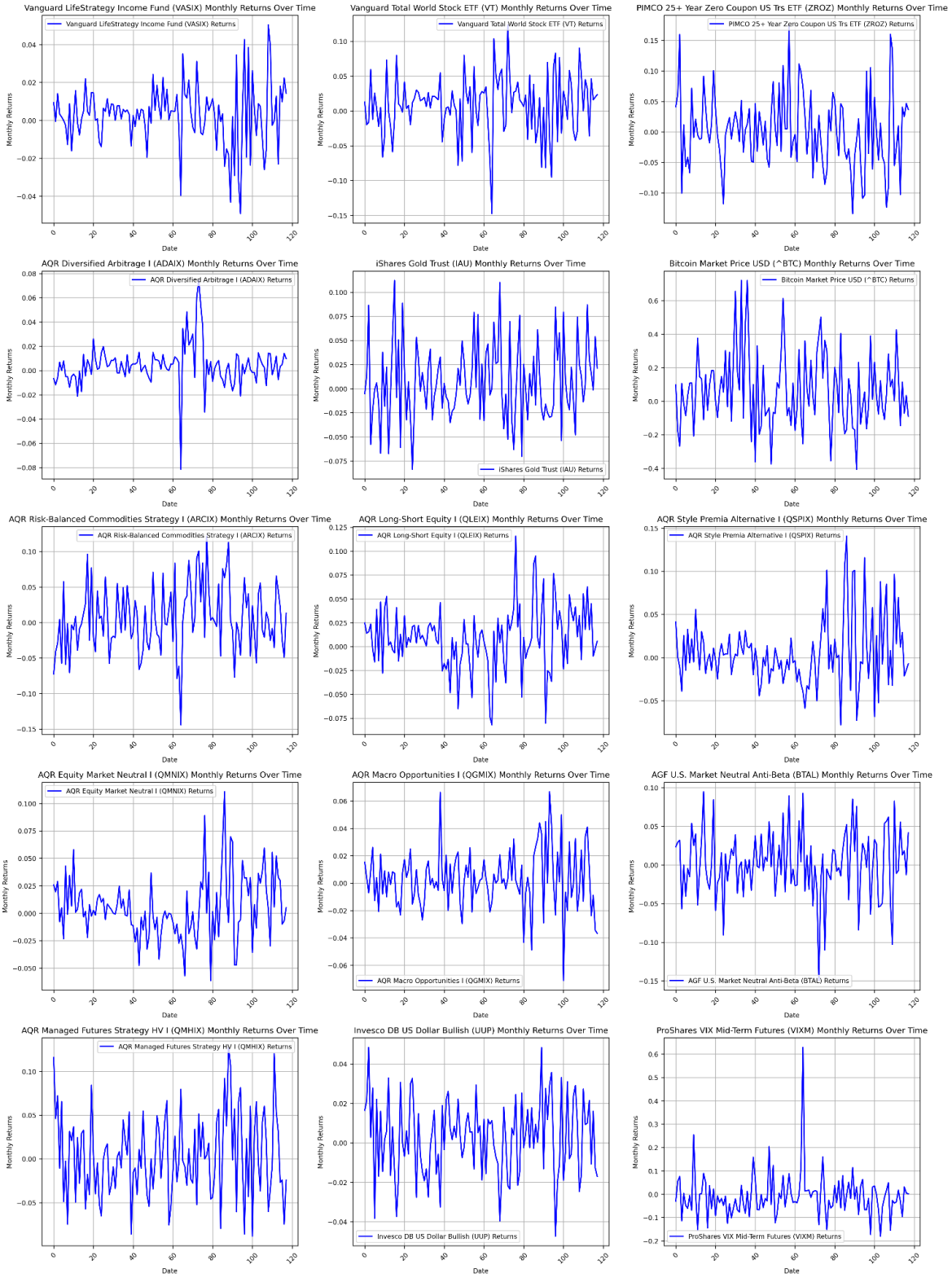


Figure 5 - Monthly Return Time Series

Box-Whisker Plots

The box-and-whisker plots in Figure 6 provide additional context on each asset's variability and distribution of returns. Numerous assets exhibit narrow interquartile ranges (IQRs) and short whiskers (e.g. ADAIX, QGMIX, UUP), indicating stable return profiles. Others that show wider IQRs consistent with their higher volatility (e.g. ZROZ, ARCIX, QMHIX, VIXM, and especially BTC).

Multiple assets with narrower IQRs still exhibit outlier returns (VASIX, in particular), so we will need to dig into outliers later because extreme events can significantly impact overall portfolio performance, and proper diversification and risk management are crucial for mitigating such risks.

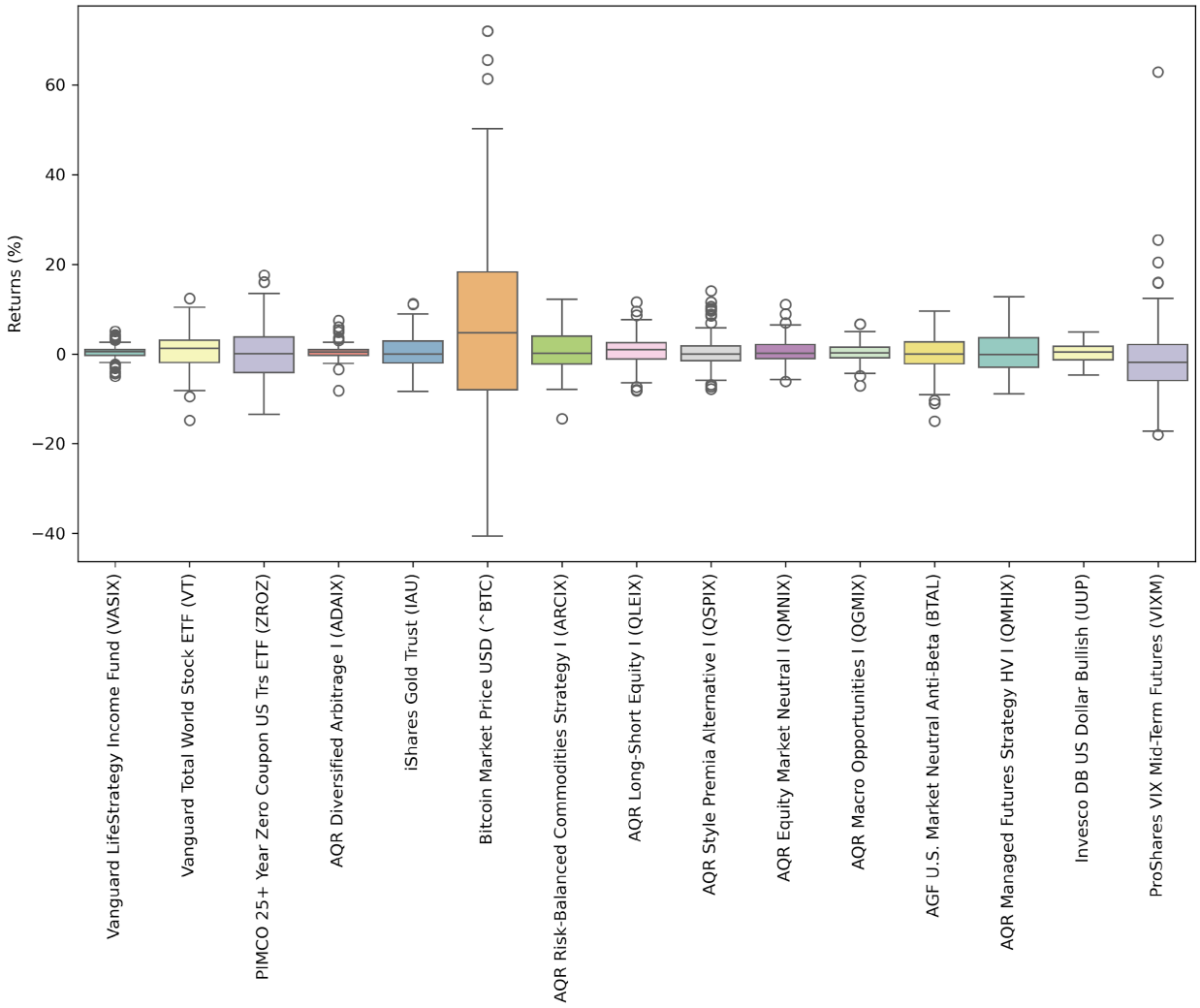


Figure 6 - Box-Whisker Plots

Outlier Analysis

Table 3 examines monthly return outliers for the benchmark asset, VASIX. It is sorted from the most positive to the most negative outlier. We see that VASIX experienced seven positive outlier months and eight negative outlier months, with five of the eight negative outliers occurring in 2022 alone. The purpose of this analysis is to see if diversification was preserved when it mattered most. To see that more clearly, you can turn your attention to the bottom of Table 3, which shows the average negative outlier return for VASIX was -3.21%. We observe that seven of the assets averaged a negative return when VASIX had a negative outlier, with three of them (VT, ZROZ, and BTC) generating a worse loss than the benchmark. However, seven other assets actually produced a positive return when VASIX was in a negative outlier condition, with the least positive among them being +2.49%. Overall, these seven assets maintained a negative correlation with VASIX when it was in an outlier condition (both positive and negative). This means a large portion of the opportunity set preserved its diversification at the most vulnerable time for conventional asset allocations like that represented by VASIX.

	Date	VASIX Return (%)	VT	ZROZ	ADAIX	IAU	^BTC	ARCIX	QLEIX	QSPIX	QMNIX	QGMIX	BTAL	QMHIX	UUP	VIXM
0	Nov-2023	5.04	9.01	16.02	1.43	2.53	8.76	0.44	3.26	0.81	0.38	-0.57	-4.84	-6.01	-2.46	-15.47
1	Nov-2022	4.26	8.28	9.96	-0.26	8.46	-16.48	6.31	5.57	1.62	3.20	-2.32	-0.29	-8.60	-4.73	-6.67
2	Dec-2023	3.95	5.15	13.44	1.39	1.27	13.01	-2.52	-1.35	-3.23	-2.98	2.42	-10.28	-3.47	-1.60	-2.67
3	Jan-2023	3.84	7.65	10.55	1.03	5.78	38.86	4.03	3.75	1.47	1.59	0.09	-6.36	-4.31	-1.08	-17.23
4	Apr-2020	3.51	10.37	2.65	3.46	6.90	36.01	0.21	1.62	-5.87	-3.22	0.66	-3.39	-0.25	-0.15	1.35
5	Jul-2022	3.45	6.98	2.16	1.38	-2.51	23.12	0.00	-2.52	-4.44	-4.73	0.00	-8.40	-6.05	1.18	-8.68
6	Nov-2020	3.11	12.37	2.72	6.07	-5.25	39.53	9.11	3.30	-0.64	-3.26	0.32	-14.96	-5.25	-2.17	-13.08
7	Apr-2024	-2.30	-3.58	-10.28	-0.75	3.07	-14.53	4.50	1.79	1.23	3.27	1.06	5.55	3.10	2.15	-3.03
8	Feb-2023	-2.36	-3.18	-6.12	0.08	-5.38	1.11	-4.73	2.48	5.80	3.25	4.99	-0.55	6.01	3.31	3.10
9	Jan-2022	-2.42	-4.58	-4.45	-1.40	-1.69	-16.74	6.31	9.50	14.08	11.07	2.66	5.23	9.20	0.94	-0.13
10	Sep-2023	-2.60	-4.25	-12.39	0.25	-4.79	3.91	-2.07	4.17	8.51	5.94	3.25	5.64	6.03	2.91	1.68
11	Jun-2022	-2.91	-8.14	-1.28	-1.11	-1.61	-40.60	-7.70	-8.02	-7.28	-4.72	4.49	7.56	5.70	2.76	3.03
12	Aug-2022	-3.14	-4.05	-5.04	1.19	-2.96	-13.66	-1.27	-2.74	-0.48	-0.83	6.68	-0.85	6.44	2.90	3.34
13	Mar-2020	-3.96	-14.76	7.33	-8.15	0.00	-25.39	-14.42	-8.21	-3.97	-1.92	-1.40	9.27	7.96	1.13	62.89
14	Apr-2022	-4.32	-8.10	-13.43	-0.83	-2.14	-16.07	0.29	3.61	9.96	6.97	3.61	8.51	10.67	4.83	11.40
15	Sep-2022	-4.92	-9.53	-10.90	-2.10	-2.87	-3.49	-4.60	-3.65	-0.72	-0.60	4.29	2.78	8.14	3.57	6.32
16	**Average Positive**	**3.88**	**8.54**	**8.21**	**2.07**	**2.45**	**20.40**	**2.51**	**1.95**	**-1.47**	**-1.29**	**0.09**	**-6.93**	**-4.85**	**-1.57**	**-8.92**
17	**Average Negative**	**-3.21**	**-6.69**	**-6.28**	**-1.42**	**-2.04**	**-13.94**	**-2.63**	**-0.12**	**3.01**	**2.49**	**3.29**	**4.79**	**7.03**	**2.72**	**9.84**
18	**Correlation**	**1.00**	**0.94**	**0.80**	**0.61**	**0.56**	**0.68**	**0.48**	**0.30**	**-0.32**	**-0.34**	**-0.64**	**-0.80**	**-0.95**	**-0.86**	**-0.59**

Table 3 - Asset Returns During Benchmark Outlier Months

Annualized Asset Risk-Return Characteristics

From here forward, we will switch to annualizing returns because that is easier for investors to relate to. Table 4 starts by displaying the compound annual growth rate (CAGR) of each asset. CAGR represents a geometric mean, as opposed to the arithmetic means we have been discussing so far. It is computed with the following formula:

$$CAGR = \left(\prod (1 + r_i)\right)^{\frac{1}{T}} - 1$$

Where:

- r_i is each monthly return.
- T is the total number of years.

Notably, while VASIX's mean monthly return only exceeded one asset, its CAGR is higher than six other assets due to a lower drag from volatility discussed above.

We also show the annualized standard deviation for each asset, which is given by the following formula:

$$\sigma_{annual} = \sigma_{monthly} \times \sqrt{12}$$

Where:

- σ_{annual} *is the annualized standard deviation* $(volatility)$.
- $\sigma_{monthly}$ *is the standard deviation of monthly returns.*
- The factor $\sqrt{12}$ accounts for the fact that there are 12 months in a year.

The table of annualized risk and return characteristics helps contextualize what standard deviation really means for an investor. While most of the CAGRs seem moderately positive, the swings in the best years versus the worst years and maximum drawdowns can be wild (even ignoring the extremes from BTC). Ten of the 14 assets have standard deviations that are twice that of VASIX. Ten have maximum drawdowns of -25% or worse (many substantially worse). This demonstrates that investing is risky business and provides important context for what we demonstrate later when combining these assets into cohesive portfolios.

Risk and Return Characteristics for All Assets

	Asset	CAGR (%)	Annualized Std Dev (%)	Best Year (%)	Worst Year (%)	Max Drawdown (%)	Sharpe Ratio	Sortino Ratio
0	Vanguard LifeStrategy Income Fund (VASIX)	3.35	5.63	12.05	-13.93	-16.72	0.26	0.10
1	Vanguard Total World Stock ETF (VT)	9.29	15.06	26.82	-18.02	-25.53	0.53	0.22
2	PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)	-0.52	21.06	24.52	-41.31	-61.73	-0.02	-0.01
3	AQR Diversified Arbitrage I (ADAIX)	5.19	5.86	25.25	-4.87	-8.15	0.55	0.20
4	iShares Gold Trust (IAU)	7.75	14.01	25.02	-10.59	-17.86	0.46	0.26
5	Bitcoin Market Price USD (^BTC)	68.78	76.62	1271.14	-71.76	-73.82	1.02	0.67
6	AQR Risk-Balanced Commodities Strategy I (ARCIX)	5.40	16.25	39.60	-19.50	-30.09	0.28	0.15
7	AQR Long-Short Equity I (QLEIX)	10.50	11.45	31.09	-16.33	-33.67	0.76	0.33
8	AQR Style Premia Alternative I (QSPIX)	5.76	13.44	30.61	-21.96	-39.58	0.33	0.20
9	AQR Equity Market Neutral I (QMNIX)	6.08	9.95	27.21	-19.52	-38.28	0.44	0.24
10	AQR Macro Opportunities I (QGMIX)	3.24	7.55	29.27	-4.55	-10.00	0.20	0.09
11	AGF U.S. Market Neutral Anti-Beta (BTAL)	1.16	14.85	20.49	-15.09	-35.41	0.02	0.01
12	AQR Managed Futures Strategy HV I (QMHIX)	2.99	16.43	49.99	-14.44	-36.26	0.14	0.08
13	Invesco DB US Dollar Bullish (UUP)	3.15	6.63	9.46	-9.12	-12.07	0.20	0.10
14	ProShares VIX Mid-Term Futures (VIXM)	-13.84	32.43	72.39	-50.05	-79.79	-0.38	-0.23

Table 4 - Asset Risk-Return Characteristics

Asset Regressions Against the Benchmark

To better understand the relationship between VASIX and the other assets, we performed an Ordinary Least Squares regression of the monthly returns of VASIX (the explanatory variable) against each asset (the response variable). Table 5 shows these results. Here, the intercept represents the annualized unexplained (arithmetic) return of an asset that is independent of the stock-bond drivers within VASIX. Beta is the coefficient that represents the sensitivity and asset has to VASIX. On this basis, VT and ZROZ have annual returns of 2.67% and-6.36%, respectively, neither of which is statistically significant at a 5% confidence level. They each have a highly statistically significant beta, consistent with the idea that they are highly related to VASIX. Eleven of the 12 other assets have positive intercepts, the lowest benign 3.83% of unexplained returns (ADAIX) that are still significant (p-value 0.0310). Five of the other 12 assets have statistically significant betas to VASIX, while six have statistically significant negative betas, which further corroborates the findings from the correlation matrix. Overall, the regression model indicates that most of the assets have independent sources of return that are economically large, if not all statistically significant. This is more promising evidence that combining them will prove fruitful.

	Asset	R2	Intercept (Annualized)	Intercept t-stat	Intercept p-value	Beta	Beta t-stat	Beta p-value
0	Vanguard Total World Stock ETF (VT)	0.64	2.67	0.90	0.3679	**2.13**	**14.25**	**0.0000**
1	PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)	0.38	-6.36	-1.18	0.2394	**2.32**	**8.50**	**0.0000**
2	AQR Diversified Arbitrage I (ADAIX)	0.15	**3.83**	**2.18**	**0.0310**	**0.41**	**4.61**	**0.0000**
3	iShares Gold Trust (IAU)	0.14	5.28	1.25	0.2155	**0.91**	**4.26**	**0.0000**
4	Bitcoin Market Price USD (^BTC)	0.08	**66.51**	**2.79**	**0.0062**	**3.92**	**3.24**	**0.0016**
5	AQR Risk-Balanced Commodities Strategy I (ARCIX)	0.06	4.10	0.80	0.4249	**0.71**	**2.75**	**0.0069**
6	AQR Long-Short Equity I (QLEIX)	0.05	**9.13**	**2.51**	**0.0133**	**0.45**	**2.42**	**0.0169**
7	AQR Style Premia Alternative I (QSPIX)	0.02	7.77	1.80	0.0744	-0.37	-1.71	0.0902
8	AQR Equity Market Neutral I (QMNIX)	0.05	**7.81**	**2.48**	**0.0145**	**-0.41**	**-2.57**	**0.0115**
9	AQR Macro Opportunities I (QGMIX)	0.12	**5.11**	**2.22**	**0.0281**	**-0.47**	**-4.07**	**0.0001**
10	AGF U.S. Market Neutral Anti-Beta (BTAL)	0.16	5.87	1.32	0.1884	**-1.04**	**-4.65**	**0.0000**
11	AQR Managed Futures Strategy HV I (QMHIX)	0.19	8.63	1.79	0.0759	**-1.26**	**-5.16**	**0.0000**
12	Invesco DB US Dollar Bullish (UUP)	0.26	**5.41**	**2.92**	**0.0042**	**-0.60**	**-6.42**	**0.0000**
13	ProShares VIX Mid-Term Futures (VIXM)	0.24	-0.51	-0.06	0.9562	**-2.80**	**-5.98**	**0.0000**

Table 5 - Asset Regressions Against the Benchmark (VASIX)

Non-Parametric Bootstraps

Table 6 shows the non-parametric bootstrap results for annualized returns and standard deviations, providing further insights into each asset's return distribution characteristics, reinforcing earlier conclusions. Across most assets, the bootstrapped confidence intervals reveal substantial return uncertainty and highlight the importance of considering the risk associated with tail events.

Although only one asset in our 118 month sample had a negative mean return (VIXM), the 95% confidence interval indicates that it is plausible for most of the assets to experience a negative average return over an approximately 10-year period.

On the right side of Table 6, we show the CAGR of each asset as well as an estimated CAGR resulting from the bootstrap process. This indicates that the bootstrap generates realistic mean estimates, but one should be cognizant of the width of the distribution and the possibility that left-tail outcomes could result in negative CAGR for prolonged periods of time. We will explore this possibility later with Monte Carlo simulations.

The formula for estimating compound annual returns (also known as geometric returns) given the arithmetic mean (μ) and the standard deviation (σ) of annual returns is derived using the following approximation:

$$\text{Estimated CAGR} \approx \mu - \frac{1}{2}\sigma^2$$

Where:

- μ is the annualized arithmetic mean of the returns.
- σ is the annualized standard deviation of the returns.

This approximation accounts for the volatility drag, which occurs because returns compound over time. The higher the volatility (σ), the more it reduces the compounded return compared to the arithmetic average return.

	Annualized Mean Estimate	Mean 95% CI Lower	Mean 95% CI Upper	Annualized Std Dev Estimate	Std Dev 95% CI Lower	Std Dev 95% CI Upper	Estimated CAGR (%)	Actual CAGR (%)
Vanguard LifeStrategy Income Fund (VASIX)	3.48	0.02	6.91	5.59	4.63	6.53	3.32	3.35
Vanguard Total World Stock ETF (VT)	9.98	0.28	19.36	14.96	12.70	17.35	8.86	9.29
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)	1.64	-11.23	14.83	20.94	18.08	23.79	-0.55	-0.52
AQR Diversified Arbitrage I (ADAIX)	5.27	1.62	8.96	5.76	4.15	7.47	5.11	5.19
iShares Gold Trust (IAU)	8.50	-0.29	17.27	13.93	12.19	15.62	7.53	7.75
Bitcoin Market Price USD (^BTC)	79.80	32.14	127.10	76.04	65.13	87.04	50.88	68.78
AQR Risk-Balanced Commodities Strategy I (ARCIX)	6.56	-3.35	16.75	16.15	14.11	18.24	5.25	5.40
AQR Long-Short Equity I (QLEIX)	10.71	3.57	17.88	11.37	9.53	13.21	10.06	10.50
AQR Style Premia Alternative I (QSPIX)	6.49	-1.77	15.08	13.31	10.96	15.61	5.61	5.76
AQR Equity Market Neutral I (QMNIX)	6.39	0.37	12.60	9.87	8.26	11.49	5.90	6.08
AQR Macro Opportunities I (QGMIX)	3.47	-1.27	8.09	7.50	6.33	8.73	3.19	3.24
AGF U.S. Market Neutral Anti-Beta (BTAL)	2.29	-7.03	11.28	14.74	12.51	17.11	1.20	1.16
AQR Managed Futures Strategy HV I (QMHIX)	4.32	-5.75	14.62	16.34	14.41	18.29	2.99	2.99
Invesco DB US Dollar Bullish (UUP)	3.31	-0.89	7.49	6.58	5.80	7.36	3.10	3.15
ProShares VIX Mid-Term Futures (VIXM)	-10.19	-29.36	11.74	31.68	21.88	44.31	-15.21	-13.84

Table 6 - Non-Parametric Bootstrap Results (10,000 Iterations)

Rolling 36-Month Annualized Means, Standard Deviations, and Risk-Adjusted Returns

The rolling 36-month means, standard deviations, and risk-adjusted returns in Figure 7 offer perspective on risk and return variability over time. There are two main takeaways:

- First, every asset experienced significant fluctuations in performance over 3-year periods; many, in fact, had negative returns at some point during the sample period. VASIX had a reasonably stable return for more than half of the time frame, but it fell negative over the latter portion. However, over half of the assets saw some of their strongest 36-month performance when

VASIX was at its lows. This highlights how certain assets can excel when others underperform over multi-year periods (as opposed to merely a monthly basis), suggesting that reasonable returns may be preserved when traditional core portfolio components (stocks and bonds) are generating lower risk adjusted returns.

- Second, standard deviation also varied over time, but to a much lesser extent than mean returns; this is important because variances combine with correlations to produce a diversification benefit in the form of a lower portfolio standard deviation. We already saw that rolling cross-correlations were low throughout the sample period (Figure 4 above). Therefore, relatively stable variances indicate that diversification may be reasonably stable, as well.

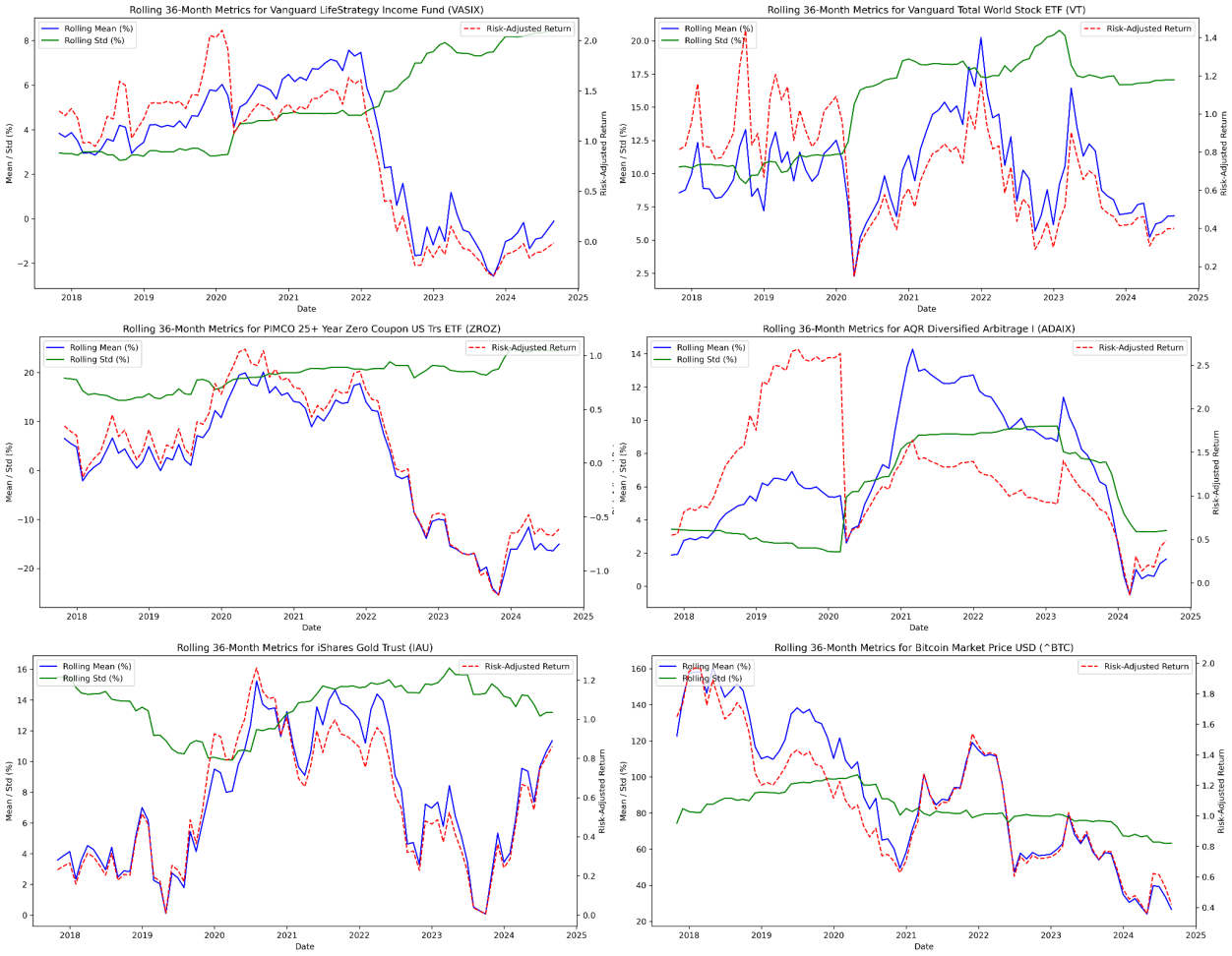


Figure 7 - Rolling 36-Month Means, Standard Deviations, and Risk-Adjusted Returns

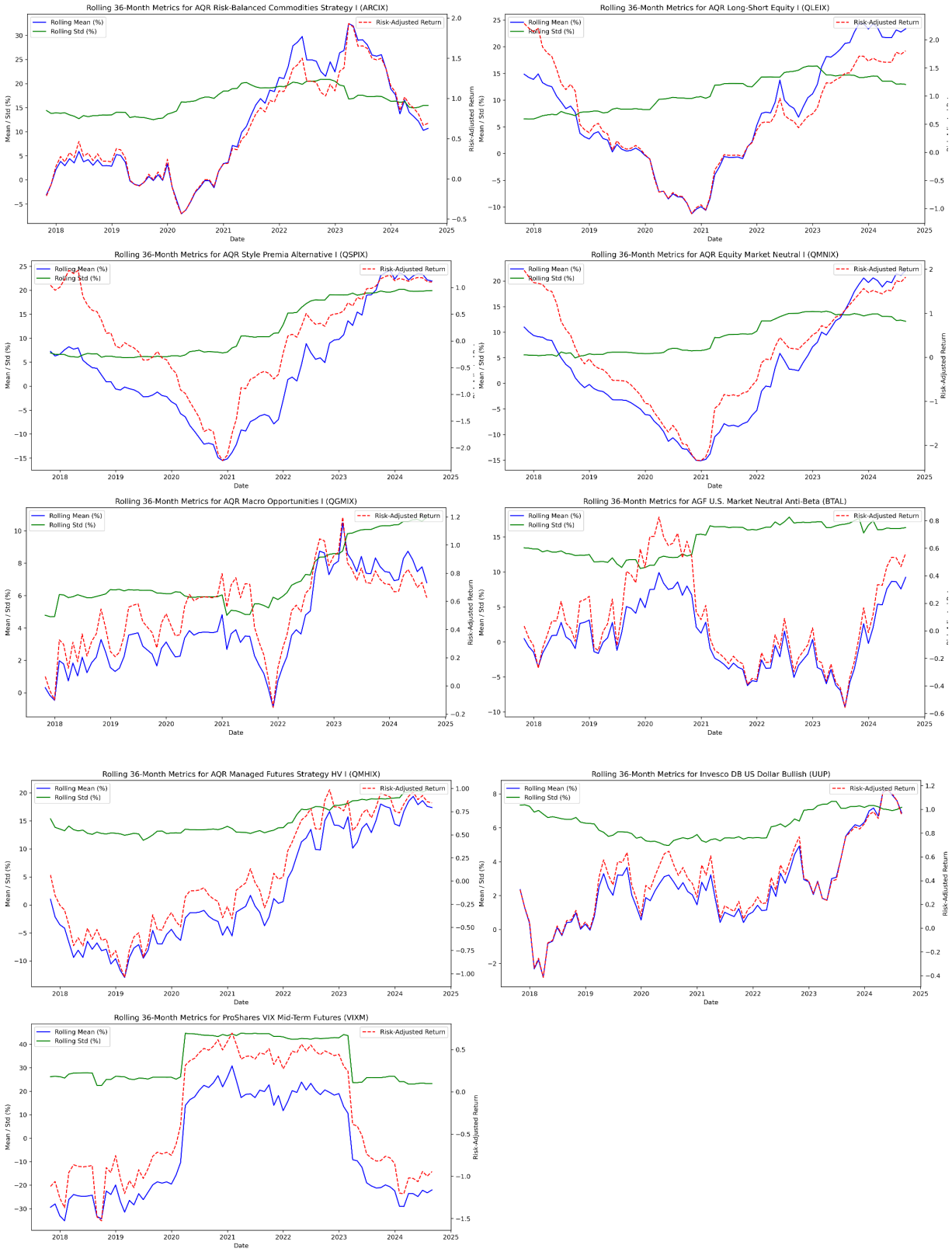


Figure 7 - Rolling 36-Month Means, Standard Deviations, and Risk-Adjusted Returns, Cont.

Raw Return Histograms with Fitted Probability Distributions

Histograms of monthly returns with fitted T, Cauchy, and normal distributions, shown in Figure 8, provide essential insights into the underlying characteristics of the asset in the opportunity set. Generally, most assets demonstrate significant deviations from normality, with higher peaks, heavier tails, and skew, suggesting that using a normal distribution to model these returns may underestimate the frequency and magnitude of extreme returns.

In summary, this analysis underscores the importance of selecting an appropriate probability distribution to model asset returns accurately. The T distribution often strikes a balance between capturing heavy tails without overestimating risk, making it a suitable candidate for modeling the returns of most assets in this study. These insights will be instrumental in informing our monte carlo analysis, ensuring that we account for non-normal return characteristics and accurately capture tail risks in our investment strategies.

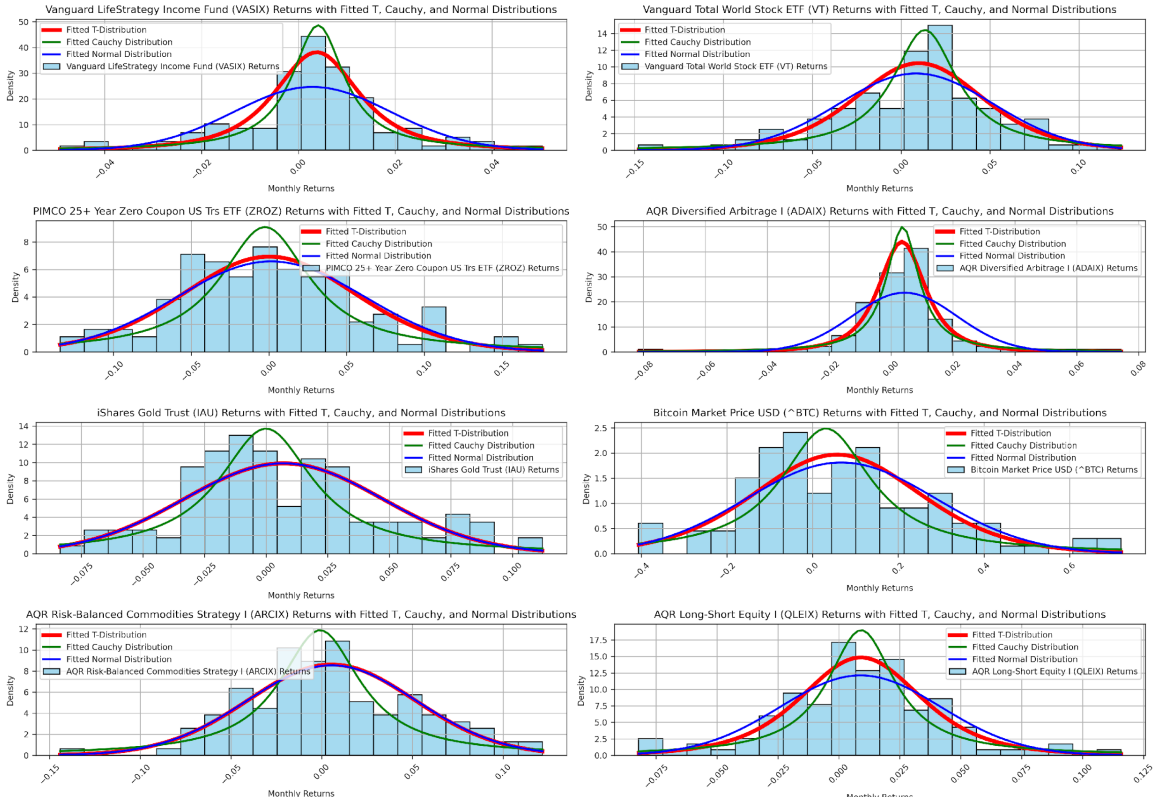


Figure 8 - Raw Return Histograms with Fitted Probability Distributions

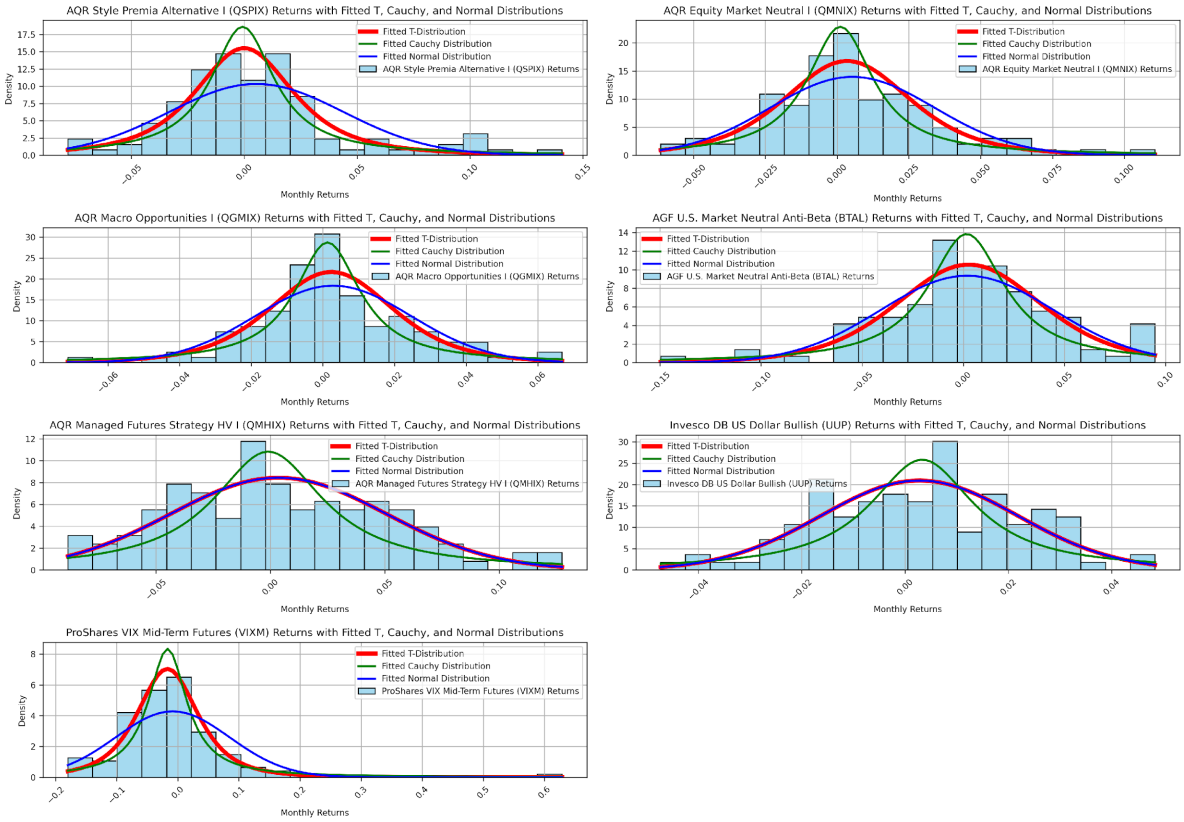


Figure 8 - Raw Return Histograms with Fitted Probability Distributions (cont.)

QQ-Plots

The QQ plots, shown in Figure 9, provide an additional validation layer for the histogram findings and fitted distributions. Across the different assets, the QQ plots consistently show that many distributions exhibit heavier tails than expected under a normal distribution, with deviations becoming especially pronounced at the extremes. This supports the earlier conclusion that a normal distribution is often insufficient for modeling financial returns, as it underestimates the likelihood of extreme events.

Overall, the QQ plots confirm that the return distributions for most assets are non-normal, with heavier tails and occasional skewness. This further highlights the importance of choosing an appropriate probability distribution—such as the T-distribution—to accurately model these asset's risk and return characteristics.

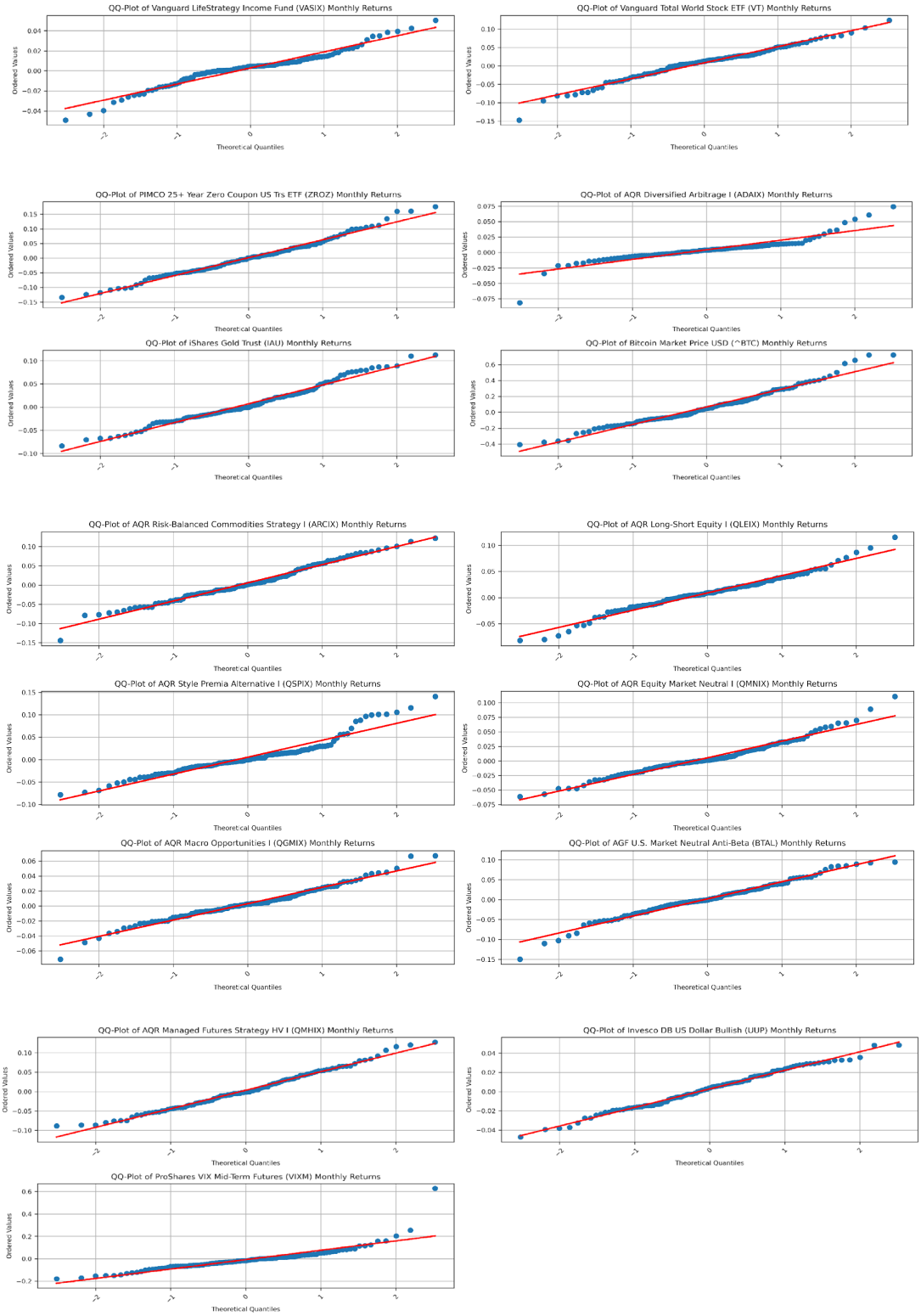


Figure 9 - QQ Plots (cont.)

Quantitative Goodness of Fit Tests: AIC, BIC, K-S

The quantitative analysis using AIC (Akaike Information Criterion), BIC (Bayesian Information Criterion), and K-S (Kolmogorov-Smirnov) tests, shown in Table 7, supports the visual interpretations derived from histograms, fitted distributions, and QQ-plots, as well as our earlier observations of skew and kurtosis. Across most assets, the T-distribution consistently emerges as the best fit according to AIC and BIC, indicating that it captures the key characteristics of the return distributions, particularly the presence of heavy tails. This is also corroborated by the K-S statistic, which generally shows the lowest values for the T distribution, indicating a better overall fit. Although we know that the assets are not normally distributed, the Cauchy distribution tends to overestimate tail risks and peakedness.

This quantitative analysis affirms that the T-distribution is often the most suitable choice for modeling the return distributions of the diverse asset set under consideration. It captures the tail behavior more accurately than the normal distribution without the extreme risk overestimation that characterizes the Cauchy distribution.

	Asset	AIC_T	AIC_Cauchy	AIC_Normal	BIC_T	BIC_Cauchy	BIC_Normal	K-S_T	K-S_Cauchy	K-S_Normal
0	Vanguard LifeStrategy Income Fund (VASIX)	**-647.000**	-639.908	-634.216	**-638.688**	-634.366	-628.675	0.058	**0.054**	0.122
1	Vanguard Total World Stock ETF (VT)	**-403.166**	-379.327	-402.084	-394.854	-373.786	**-396.542**	**0.066**	0.072	0.089
2	PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)	-321.459	-288.056	**-322.976**	-313.147	-282.515	**-317.435**	**0.039**	0.083	0.052
3	AQR Diversified Arbitrage I (ADAIX)	**-679.596**	-667.710	-624.802	**-671.284**	-662.168	-619.261	**0.047**	0.070	0.169
4	iShares Gold Trust (IAU)	-417.207	-382.092	**-419.207**	-408.895	-376.550	**-413.665**	0.078	0.094	**0.078**
5	Bitcoin Market Price USD (^BTC)	-17.637	15.443	**-18.210**	-9.325	20.985	**-12.669**	**0.071**	0.097	0.076
6	AQR Risk-Balanced Commodities Strategy I (ARCIX)	-382.235	-345.458	**-384.209**	-373.923	-339.916	**-378.668**	**0.066**	0.097	0.067
7	AQR Long-Short Equity I (QLEIX)	**-471.236**	-448.901	-466.863	**-462.924**	-443.360	-461.322	**0.034**	0.059	0.064
8	AQR Style Premia Alternative I (QSPIX)	**-444.950**	-432.415	-429.054	**-436.638**	-426.874	-423.513	**0.055**	0.084	0.134
9	AQR Equity Market Neutral I (QMNIX)	**-503.627**	-483.176	-499.914	**-495.315**	-477.635	-494.372	**0.063**	0.077	0.101
10	AQR Macro Opportunities I (QGMIX)	**-567.909**	-544.492	-564.954	**-559.597**	-538.950	-559.412	0.068	**0.067**	0.084
11	AGF U.S. Market Neutral Anti-Beta (BTAL)	**-406.536**	-379.027	-405.510	-398.224	-373.485	**-399.969**	**0.042**	0.077	0.060
12	AQR Managed Futures Strategy HV I (QMHIX)	-379.619	-334.573	**-381.619**	-371.307	-329.032	**-376.078**	**0.059**	0.103	0.059
13	Invesco DB US Dollar Bullish (UUP)	-593.853	-545.432	**-595.853**	-585.541	-539.890	**-590.312**	**0.054**	0.099	0.054
14	ProShares VIX Mid-Term Futures (VIXM)	**-270.022**	-252.818	-221.081	**-261.710**	-247.276	-215.539	**0.048**	0.073	0.140

Table 7 - Quantitative Goodness of Fit Tests (AIC, BIC, K-S)

Monte Carlo Simulations

We conclude our exploratory data analysis with monte carlo simulations that consider 50th percentile (Table 8), 10th percentile (Table 9), and 90th percentile (Table 10) paths for each asset over a 36-month time period that is more relevant to a risk-averse investor than a 10-year horizon. We run 10,000 simulations using a T-distribution to better capture the extremity of the tails than if we had chosen to use a Normal distribution.

Important takeaways from this exercise are that 50th percentile outcomes still come with severe drawdowns within the 36-month period for any given asset. Every asset can experience a negative CAGR over 3-years. Even the conservative benchmark asset, VASIX, can produce a loss over a 3-year period in a 10th percentile episode–this very outcome was observed in our sample as shown in the rolling 36-month plot in Figure 6 above. Ultra-volatile assets like BTC have a real risk of a total wipeout–this does not mean it cannot be a component of a diversified portfolio, but that its statistical history comes with a starker warning than perhaps many commentators and enthusiasts understand. Finally, due to their volatility, even assets that have negative expected CAGR in a 3-year period (ZROZ, VIXM) can still have an expectation of strong performance when the right tail kicks in like in a 90th percentile outcome; this may be a welcome outcome if other uncorrelated assets are simultaneously experiencing a 10th percentile result. We will repeat this monte carlo simulation for the optimized portfolios later in this report, so the results for the individual assets can serve as helpful context when evaluating the benefit of diversification.

Asset	Portfolio End Balance ($)	Annual Compounded Return (%)	Annualized Volatility (%)	Maximum Drawdown (%)
Vanguard LifeStrategy Income Fund (VASIX)	11,015	3.27	6.88	-8.18
Vanguard Total World Stock ETF (VT)	12,825	8.65	18.40	-20.64
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)	9,464	-1.82	25.73	-37.05
AQR Diversified Arbitrage I (ADAIX)	11,612	5.11	7.16	-7.51
iShares Gold Trust (IAU)	12,310	7.17	17.11	-19.83
Bitcoin Market Price USD (^BTC)	23,976	33.84	93.60	-74.97
AQR Risk-Balanced Commodities Strategy I (ARCIX)	11,458	4.64	19.85	-25.07
AQR Long-Short Equity I (QLEIX)	13,357	10.13	13.98	-14.22
AQR Style Premia Alternative I (QSPIX)	11,650	5.22	16.41	-20.20
AQR Equity Market Neutral I (QMNIX)	11,843	5.80	12.16	-14.04
AQR Macro Opportunities I (QGMIX)	10,955	3.09	9.23	-11.74
AGF U.S. Market Neutral Anti-Beta (BTAL)	10,170	0.56	18.14	-26.03
AQR Managed Futures Strategy HV I (QMHIX)	10,683	2.23	20.07	-27.10
Invesco DB US Dollar Bullish (UUP)	10,940	3.04	8.10	-10.11
ProShares VIX Mid-Term Futures (VIXM)	5,676	-17.20	39.62	-62.35

Table 8 - 50th Percentile Monte Carlo Simulations Based on T-Distributions (36-Mo, 10,000 iterations)

Asset	Portfolio End Balance ($)	Annual Compounded Return (%)	Annualized Volatility (%)	Maximum Drawdown (%)
Vanguard LifeStrategy Income Fund (VASIX)	9,378	-2.12	5.55	-15.54
Vanguard Total World Stock ETF (VT)	8,309	-5.99	14.84	-37.31
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)	5,105	-20.08	20.75	-59.63
AQR Diversified Arbitrage I (ADAIX)	9,823	-0.59	5.77	-14.40
iShares Gold Trust (IAU)	8,226	-6.30	13.80	-35.93
Bitcoin Market Price USD (^BTC)	309	-68.63	75.48	-99.25
AQR Risk-Balanced Commodities Strategy I (ARCIX)	7,158	-10.55	16.01	-43.87
AQR Long-Short Equity I (QLEIX)	9,639	-1.22	11.28	-26.57
AQR Style Premia Alternative I (QSPIX)	7,912	-7.51	13.24	-36.35
AQR Equity Market Neutral I (QMNIX)	8,914	-3.76	9.80	-26.05
AQR Macro Opportunities I (QGMIX)	8,829	-4.07	7.44	-21.89
AGF U.S. Market Neutral Anti-Beta (BTAL)	6,612	-12.88	14.63	-44.81
AQR Managed Futures Strategy HV I (QMHIX)	6,629	-12.81	16.18	-46.65
Invesco DB US Dollar Bullish (UUP)	9,052	-3.27	6.53	-19.05
ProShares VIX Mid-Term Futures (VIXM)	2,079	-40.76	31.95	-83.87

Table 9 - 10th Monte Carlo Simulations Based on T-Distributions (10,000 iterations)

Asset	Portfolio End Balance ($)	Annual Compounded Return (%)	Annualized Volatility (%)	Maximum Drawdown (%)
Vanguard LifeStrategy Income Fund (VASIX)	12,887	8.82	8.71	-4.41
Vanguard Total World Stock ETF (VT)	19,493	24.92	23.29	-11.41
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)	17,103	19.59	32.57	-20.37
AQR Diversified Arbitrage I (ADAIX)	13,669	10.98	9.06	-4.08
iShares Gold Trust (IAU)	18,175	22.04	21.66	-10.92
Bitcoin Market Price USD (^BTC)	243,477	189.84	118.48	-46.46
AQR Risk-Balanced Commodities Strategy I (ARCIX)	18,026	21.70	25.13	-13.68
AQR Long-Short Equity I (QLEIX)	18,340	22.41	17.70	-7.79
AQR Style Premia Alternative I (QSPIX)	16,936	19.20	20.78	-11.01
AQR Equity Market Neutral I (QMNIX)	15,622	16.03	15.39	-7.66
AQR Macro Opportunities I (QGMIX)	13,524	10.59	11.68	-6.30
AGF U.S. Market Neutral Anti-Beta (BTAL)	15,407	15.50	22.96	-14.05
AQR Managed Futures Strategy HV I (QMHIX)	16,909	19.14	25.40	-14.71
Invesco DB US Dollar Bullish (UUP)	13,159	9.58	10.25	-5.43
ProShares VIX Mid-Term Futures (VIXM)	14,476	13.12	50.15	-37.16

Table 10 - 90th Monte Carlo Simulations Based on T-Distributions (10,000 iterations)

Model Selection

Risk Parity, Maximum Diversification, and Minimum Variance: An Analytic Perspective by Clarke, de Silva, and Thorley (2013) discusses three portfolio optimization methods that use covariance matrices to construct portfolios which balance risk across assets without explicitly considering expected returns. We describe these three models in this section and test them in the next section.

Before presenting each method, it is worth noting two things about them:

- All the methods assume that the covariance matrix contains all the necessary information to describe the risk of the assets. Implicitly, this assumes that the returns are normally distributed and that relationships between assets are linear. We have already established that the probability distributions are non-normal. This suggests that more sophisticated methods of optimization may better account for these properties. Nevertheless, we believe this exercise is worthwhile to at least establish a baseline of performance and to see if violations of these assumptions appear to manifest in breakdowns of the portfolios as we noted with VASIX.

- Furthermore, these risk-based optimizations do not make assumptions about mean returns. There are two fair perspectives about this. One is that mean returns have such a wide confidence interval that is not worth weighting assets based on them. The other is that an investor should have an informed estimate of mean returns and should incorporate these forecasts into an optimization. We will see later if excluding means from consideration has an obviously detrimental impact.

Risk Parity Optimization

The principle behind risk parity is to weight assets in a portfolio so that each one contributes equally to the overall portfolio risk as measured by standard deviation. In mathematical terms, asset weights are inversely proportional to their volatility and adjusted for correlations between assets. Assuming no assets in the investable set have exceptionally low volatility, this approach avoids high concentrations in any one asset and spreads risk evenly across the portfolio, including all assets in the investable set.

Objective Function: The goal of Risk Parity is to equalize the risk contributions of each asset to the total portfolio risk. The total portfolio risk is defined as:

$$\sigma_P^2 = \sum_i \sum_j w_i w_j \sigma_{ij}$$

Where:

- w_i and w_j are the weights of assets *i* and *j.*
- σ_{ij} represents the covariance between assets *i* and *j.*

This formula calculates the portfolio's total variance, taking into account the weighted covariances between all pairs of assets in the portfolio. The portfolio is in risk parity when each asset's risk contribution is the same. This means that:

$$w_i \times Marginal\ Risk\ Contribution = \frac{\sigma p}{N}$$

Where:

- *N* is the total number of assets.

Optimization constraints: Risk Parity portfolios are long-only, meaning weights are non-negative (i.e., $w_i \geq 0$ for all *i*), and the sum of the weights must equal 1.

$$\sum_{i}^{\square} \square w_i = 1$$

This constraint ensures that the portfolio uses the full capital allocation without shorting any assets.

Maximum Diversification Optimization

Maximum diversification aims to maximize the diversification ratio, which measures how much diversification is gained by comparing the weighted sum of individual asset volatilities (standard deviations)–for the assets that are given a positive weight–to the overall portfolio volatility. A higher ratio implies better diversification.

Objective Function: The objective of Maximum Diversification is to maximize the diversification ratio, which is defined as:

$$D_p = \frac{\sum_i^{\square} \square w_i \sigma_i}{\sigma_p}$$

Where:

- *w* is the vector of portfolio weights
- σ_i is the volatility of the asset i
- σ_p is the portfolio's total volatility

This ratio measures how diversified the portfolio is relative to the weighted average volatilities of its constituent assets.

Optimization constraints: Our Maximum Diversification optimizations are constrained to be long-only, meaning weights are non-negative (i.e., $w_i \geq 0$ for all *i*), and the sum of the weights must equal 1.

$$\sum_{i} w_i = 1$$

This constraint ensures that the portfolio uses the full capital allocation without shorting any assets.

Minimum Variance Optimization

The minimum variance portfolio seeks to minimize the portfolio's standard deviation without considering expected returns. The optimization process aims to combine assets such that the resulting weighted variance is as small as possible. Assets with lower variance (less risk) tend to receive higher weights, though this is mediated by correlations with other assets. The portfolio can become relatively concentrated, often including a smaller subset of the investable assets compared to risk parity.

Objective Function: The Minimum Variance method minimizes expected portfolio variance without considering expected returns. The objective function is:

$$Minimize\ \sigma^2 p = w^{\top} \sum w$$

- $w^{\top}$: Vector of asset weights in the portfolio.
- $\sum$: Asset covariance matrix.

The objective here is purely to reduce the overall risk of the portfolio, making the minimum variance portfolio the leftmost point on the efficient frontier, representing the lowest possible risk (defined as standard deviation, the square root of variance) for a given set of assets.

Optimization constraints: Our Minimum Variance portfolios are constrained to be long-only, meaning weights are non-negative (i.e., $w_i \geq 0$ for all i), and the sum of the weights must equal 1.

$$\sum_{i} w_i = 1$$

This constraint ensures that the portfolio uses the full capital allocation without shorting any assets.

Model Analysis

Having run the optimizations described above on our dataset, we now examine the findings. Remember that VASIX is the benchmark asset and is not included in the investable set of the optimizations.

Portfolio Weights

Table 11 shows the weight of each asset in each of the optimized portfolios. All three portfolios allocate at least 20% to one asset (for reference, if the portfolios were equally weighted, each asset would have roughly a 7% weight). Risk Parity retains and invests in all the assets, whereas Minimum Variance and Maximum Diversification each exclude four assets (three of which are in common). Risk Parity and Maximum Diversification both allocate some to BTC, but less than 1% in both cases due to its high volatility.

Asset	Risk Parity (%)	Min Variance (%)	Max Diversification (%)
Vanguard Total World Stock ETF (VT)	11.62	15.84	23.15
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)	4.75	2.01	3.38
AQR Diversified Arbitrage I (ADAIX)	15.04	22.46	8.49
iShares Gold Trust (IAU)	4.05	3.29	0.00
Bitcoin Market Price USD (^BTC)	0.92	0.00	0.46
AQR Risk-Balanced Commodities Strategy I (ARCIX)	4.27	0.31	4.08
AQR Long-Short Equity I (QLEIX)	4.58	0.00	0.00
AQR Style Premia Alternative I (QSPIX)	3.82	0.00	0.00
AQR Equity Market Neutral I (QMNIX)	5.59	9.09	9.16
AQR Macro Opportunities I (QGMIX)	8.89	7.80	8.07
AGF U.S. Market Neutral Anti-Beta (BTAL)	6.13	6.69	8.73
AQR Managed Futures Strategy HV I (QMHIX)	2.68	0.00	0.00
Invesco DB US Dollar Bullish (UUP)	22.30	27.87	26.54
ProShares VIX Mid-Term Futures (VIXM)	5.36	4.64	7.92

Table 11 - Optimized Portfolio Portfolio Weights Table

Risk Contribution

Table 12 shows how Risk Parity achieves an equal contribution of risk for all 14 assets (approximately 7.1% each). By contrast, due to some higher allocations to certain assets and exclusion of other assets altogether, Minimum Variance and Maximum Diversification have rather wide-ranging risk contributions across their holdings, with some assets contributing 20%+ and others contributing less than 5%.

Asset	Risk Parity (%)	Min Variance (%)	Max Diversification (%)
Vanguard Total World Stock ETF (VT)	7.19	16.90	27.12
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)	7.11	1.94	5.54
AQR Diversified Arbitrage I (ADAIX)	7.09	22.05	3.87
iShares Gold Trust (IAU)	7.14	3.30	0.00
Bitcoin Market Price USD (^BTC)	7.10	0.00	2.74
AQR Risk-Balanced Commodities Strategy I (ARCIX)	7.18	0.29	5.16
AQR Long-Short Equity I (QLEIX)	7.15	0.00	0.00
AQR Style Premia Alternative I (QSPIX)	7.16	0.00	0.00
AQR Equity Market Neutral I (QMNIX)	7.14	9.25	7.09
AQR Macro Opportunities I (QGMIX)	7.16	7.90	4.74
AGF U.S. Market Neutral Anti-Beta (BTAL)	7.11	6.49	10.08
AQR Managed Futures Strategy HV I (QMHIX)	7.17	0.00	0.00
Invesco DB US Dollar Bullish (UUP)	7.16	27.48	13.68
ProShares VIX Mid-Term Futures (VIXM)	7.14	4.40	19.98

Table 12 - Optimized Portfolio Risk Contribution Table

Annual Returns

Table 13 displays the annual returns of each portfolio as well as the benchmark asset, VASIX. Perhaps the biggest observation is the 2022 return, when VASIX lost almost 14%. In 2022, all three optimized portfolios generated low-to-mid positive single digit returns. This is consistent with our earlier findings of stable cross-correlations of the assets, relatively stable variances, and preserving their correlations when VASIX was experiencing negative outlier months.

Some other observations are that none of the optimized portfolios ever returned 10% or more in any one year, while VASIX did once. However, VASIX experienced calendar year losses twice, whereas Risk Parity only did once (with a slight -0.34% return in 2018), Minimum Variance did once (-0.03% in 2018), while Maximum Diversification never had a calendar year loss.

Year End Date	Risk Parity (%)	Minimum Variance (%)	Maximum Diversification (%)	VASIX (Benchmark) (%)
2014-12-31	2.11	1.77	1.77	0.89
2015-12-31	1.36	1.39	1.29	0.23
2016-12-31	4.17	3.82	3.35	4.60
2017-12-31	3.51	-0.03	0.39	6.97
2018-12-31	-0.34	2.60	1.60	-1.06
2019-12-31	7.19	6.64	7.48	12.05
2020-12-31	9.11	8.56	9.61	9.14
2021-12-31	7.99	5.97	7.47	1.92
2022-12-31	5.71	4.01	3.29	-13.93
2023-12-31	4.62	4.07	3.03	9.48
2024-12-31	7.27	6.92	6.94	5.21

Table 13 - Optimized Portfolio Annual Return Table Versus Benchmark

Performance Summary

Table 14 shows that all three optimizations generated a CAGR above VASIX (the slightly higher return from Risk Parity relative to the other two optimizations is due to its larger allocation to BTC). However, this superior growth rate did not come with elevated risks. In fact, the standard deviation of all three optimized portfolios was less than 3% per year whereas VASIX had more than a 5.5% standard deviation–this was apparent from the narrow distributions of calendar year returns in Table 12 for the optimizations versus VASIX. Lastly, none of the portfolios experienced a maximum drawdown below -3% whereas VASIX saw nearly a 17% drawdown.

	Risk Parity	Minimum Variance	Maximum Diversification	VASIX (Benchmark)
CAGR (%)	5.38	4.66	4.71	3.38
Standard Deviation (%)	2.93	2.35	2.61	5.63
Maximum Drawdown (%)	-2.87	-2.21	-2.77	-16.72

Table 14 - Optimized Portfolio Performance Summary Table Versus Benchmark

Optimized Portfolio Rolling Performance

Earlier, we saw that individual assets had big swings in their rolling 36-month mean return. In Figure 10, we observe that while VASIX experienced a large drop off in its rolling 36-month return, which was attended by a near doubling in its rolling 36-month standard deviation, the optimized portfolios did not experience these same deleterious effects.

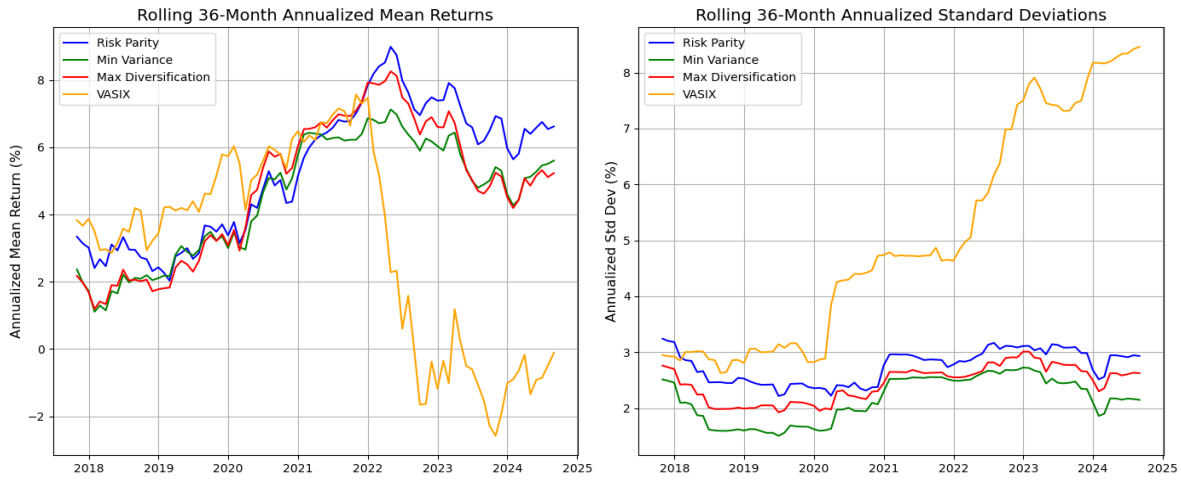


Figure 10 - Optimized Portfolio Rolling 36-month Mean and Standard Deviations

Regression Analysis

Table 15 shows the results of the same method of regression that was performed on the individual assets. All three portfolios have a small positive beta to VASIX due to the inclusion of VT and ZROZ in them. However, the bigger takeaway is that they have an intercept that is 90% plus of their arithmetic return, meaning that nearly all of their return was from sources other than the exposures within the benchmark (intercepts are also arithmetic, so we can do an apples to apples comparison). Correspondingly, they all have coefficients of determination (R-squared) near zero, supporting the idea that the return drivers come from exposures that are diversified well beyond that of bonds and stocks.

	Portfolio	R2	Annualized Arithmetic Return (%)	Intercept (Annualized)	Intercept t-stat	Intercept p-value	Beta	Beta t-stat	Beta p-value
0	Risk Parity	0.03	5.25	**4.96**	**5.26**	**0.00**	0.08	1.73	0.09
1	Minimum Variance	0.03	4.56	**4.33**	**5.74**	**0.00**	0.07	1.75	0.08
2	Maximum Diversification	0.05	4.60	**4.24**	**5.13**	**0.00**	**0.11**	**2.53**	**0.01**

Table 15 - Optimized Portfolio Performance Regression Against Benchmark

Optimized Portfolio Summary Statistics

The mean, standard deviation and monthly average returns for the optimized portfolios in Table 16 are unsurprising, given the performance described so far. The main observation here is that the optimized portfolios had right-skew, whereas VASIX had negative skew. This indicates that negative outcomes were successfully truncated and that there was a higher frequency of high one-month returns. This is evident in the smaller magnitude of the minimum returns versus the maximum returns for the optimized portfolios.

	Count	Correlation with VASIX	Mean	Std Dev	Min	25%	Median	75%	Max	Skewness	Kurtosis
Risk Parity	118	0.16	0.44	0.85	-1.60	-0.03	0.38	0.89	3.01	0.40	0.69
Minimum Variance	118	0.16	0.38	0.68	-1.41	-0.02	0.36	0.77	2.33	0.45	1.33
Maximum Diversification	118	0.23	0.38	0.75	-1.54	-0.02	0.38	0.84	2.49	0.19	0.70
VASIX (Benchmark)	118	1.00	0.29	1.63	-4.92	-0.29	0.47	0.97	5.04	-0.24	1.59

Table 16 - Optimized Portfolio Summary Statistics

Optimized Portfolio Goodness-of-Fit

The goodness-of-fit tests for the portfolios indicates that T-distributions continue to be the best probability distribution to describe them (Table 17).

Portfolio	AIC_T	AIC_Cauchy	AIC_Normal	BIC_T	BIC_Cauchy	BIC_Normal	K-S_T	K-S_Cauchy	K-S_Normal
Risk Parity	**-788.471**	-761.207	-788.169	-780.159	-755.665	**-782.628**	**0.052**	0.075	0.062
Minimum Variance	**-845.925**	-824.570	-840.664	**-837.613**	-819.028	-835.123	**0.045**	0.070	0.074
Maximum Diversification	**-816.732**	-793.282	-815.832	-808.420	-787.740	**-810.291**	**0.048**	0.065	0.068
VASIX (Benchmark)	**-647.000**	-639.908	-634.216	**-638.688**	-634.366	-628.675	0.058	**0.054**	0.122

Table 17 - Optimized Portfolio Quantitative Goodness of Fit Tests (AIC, BIC, K-S)

Optimized Portfolio Monte Carlo Analysis

When we run the portfolios through the 36-month monte carlo simulations, modeled as T-distributions, we see that all three optimized portfolios had 10th percentile CAGR outcomes of 2%+ over 3-year periods, which is similar to the 3-year mean return seen in Figure 10 around the year 2018 (Table 18). This compares to a 10th percentile return for VASIX of -2%, which was observed in the 3-years ending just before 2024. The optimized portfolios did not quite experience 10th percentile drawdowns during the sample period, while VASIX did surpass its 10th percentile drawdown estimate slightly.

	Portfolio	Percentile	Portfolio End Balance ($)	Annual Compounded Return (%)	Annualized Volatility (%)	Maximum Drawdown (%)
0	Risk Parity	90th Percentile	12,672	8.21	4.54	-1.44
1	Risk Parity	50th Percentile	11,681	5.32	3.58	-2.74
2	Risk Parity	10th Percentile	10,747	2.43	2.89	-5.38
3	Minimum Variance	90th Percentile	12,222	6.92	3.63	-1.09
4	Minimum Variance	50th Percentile	11,451	4.62	2.87	-2.09
5	Minimum Variance	10th Percentile	10,711	2.32	2.31	-4.13
6	Maximum Diversification	90th Percentile	12,324	7.21	4.03	-1.29
7	Maximum Diversification	50th Percentile	11,463	4.66	3.19	-2.46
8	Maximum Diversification	10th Percentile	10,643	2.10	2.57	-4.83
9	VASIX (Benchmark)	90th Percentile	12,887	8.82	8.71	-4.41
10	VASIX (Benchmark)	50th Percentile	11,015	3.27	6.88	-8.18
11	VASIX (Benchmark)	10th Percentile	9,378	-2.12	5.55	-15.54

Table 18 - Optimized Portfolio Monte Carlo Analysis (36-Months, 10,000 Iterations)

Ex-Ante vs Ex-Post Portfolios

One critical aspect to discuss is that these portfolio results are ex-post, meaning that the optimized weights were determined based on the full sample of asset correlations and standard deviations which could not have been known with certainty in advance. We did this to first illustrate the concept with the largest time frame of results. However, to have confidence that these optimizations can be used effectively in an ex-ante context (meaning, setting weights based on past characteristics and then assessing the subsequent performance when the future correlations and standard deviations are unknown), we also conducted rolling optimizations based on prior 12-month correlations and variances. For example, the rolling portfolio weights assigned in November of 2015 were based on the prior 12-month covariance data from November, 2014 through October, 2015. Then the lookback rolls forward by a month such that the December, 2014 weights are based on the 12-months ending in November, 2014. This had the effect of shortening the evaluation period to 106 months (just under nine years).

For brevity, we are only displaying the annual returns and performance summary table (Table 19), but the results of the other exhibits are consistent with these ex-ante findings, which are that all three optimized portfolios show slightly better performance in every respect (CAGR, standard deviation, drawdowns, and worst calendar years).

First date for rolling portfolio returns: 2015-11-30

Annual Return Table (%):

Date	Risk Parity (%)	Minimum Variance (%)	Maximum Diversification (%)	VASIX (Benchmark) (%)
2015-12-31	2.50	2.71	2.67	0.96
2016-12-31	2.95	2.32	1.87	3.13
2017-12-31	3.97	0.88	1.35	7.06
2018-12-31	0.05	2.97	1.91	-0.37
2019-12-31	7.33	6.42	7.19	11.35
2020-12-31	6.10	6.16	7.48	8.35
2021-12-31	9.23	7.09	8.05	2.80
2022-12-31	7.79	5.70	5.70	-11.94
2023-12-31	4.13	3.46	1.99	3.28
2024-12-31	7.25	6.68	6.60	7.83

Table 19 - Rolling Optimized Portfolio Annual Return

Performance Metrics Table:

	CAGR (%)	Standard Deviation (%)	Maximum Drawdown (%)
Risk Parity	5.59	2.81	-1.62
Minimum Variance	4.79	2.20	-1.52
Maximum Diversification	4.88	2.50	-2.00
VASIX (Benchmark)	3.55	5.85	-16.72

Table 20 - Performance Metrics Table

Conclusion and Recommendations

Our analysis demonstrates that the diversification opportunities available within the asset set offer substantial potential for optimizing portfolios that meet clients' needs for consistent returns while mitigating risk. The Risk Parity approach, in particular, stands out due to its balanced risk distribution and ability to include a wide range of assets without over-concentration. Based on our findings, we propose the following recommendations and next steps to further refine and enhance portfolio strategies:

1. **Extend Data Collection**: Our analysis used a common time frame during which all assets were publicly available as mutual funds or ETFs. While this period included a variety of macroeconomic conditions, it does not capture the full range of market cycles. Expanding the dataset to include additional periods and a broader range of macroeconomic conditions could improve the model's robustness. This can be achieved through:
    - Including funds with longer historical records.
    - Using similar funds or proxies for periods before specific assets were available.
    - Incorporating published indexes that represent these products or their proxies.
    - Creating regression-based approximations using risk factor and macroeconomic data.
    - Simulating price histories to preserve variance and correlations across assets.
2. **Increase Sample Size**: Expanding the dataset would effectively increase the sample size, which could provide more reliable insights into the stability of covariance structures and other statistical properties.

3. **Leverage Machine Learning for Asset Clustering**: Employ machine learning techniques to cluster assets based on their unique risk-return characteristics. Clustering could simplify portfolio construction by reducing redundancy and offering a representative set of assets that capture distinct risk factors, particularly for the Risk Parity approach.
4. **Broaden Goodness-of-Fit Testing**: Extend the goodness-of-fit analysis by incorporating additional tests, such as Anderson-Darling, Shapiro-Wilk, D'Agostino's K-squared, and Jarque-Bera, to assess skewness and kurtosis more comprehensively. Additionally, consider alternative probability distributions to enhance the accuracy of the model.
5. **Improve Monte Carlo Simulations**: Conduct more precise Monte Carlo simulations over multiple time periods, using parametric simulations informed by enhanced goodness-of-fit findings. This will provide deeper insights into asset and portfolio performance under various stress-testing scenarios.
6. **Conduct Risk Factor Analysis**: Perform detailed regression analyses of assets against macroeconomic and financial market factors to isolate key performance drivers. This could also help in constructing synthetic returns for assets with limited historical data, filling gaps in time series data. Further, regression results can inform stress tests for hypothetical financial market scenarios, providing a broader view of potential risks.
7. **Evaluate Additional Assets**: Explore other investment products that may provide exposure to uncorrelated returns. Consider newer funds that may not have been included in this study due to the limited historical sample. Broader asset selection could further enhance diversification opportunities.
8. **Revisit Other Optimization Methods**: With additional data and deeper analysis, it may be worth reevaluating other portfolio optimization models, including non-linear methods not discussed in

this study. These methods could offer superior performance in addressing non-normal return distributions.

Overall, this study demonstrates the efficacy of diversified portfolio construction strategies that extend beyond traditional stock-bond allocations using liquid investment products. While the Risk Parity approach offers a solid foundation for portfolio optimization, we recommend further exploration of the above strategies to enhance the robustness and understanding of portfolio management.

References

Agresti, A., & Kateri, M. (2022). Foundations of statistics for data scientists: With R and Python. *CRC Press, Taylor & Francis Group*.

OpenAI. (2024). ChatGPT (GPT-4o version) [Large language model]. https://chatgpt.com/

Clarke, R., de Silva, H., & Thorley, S. (2023). Risk parity, maximum diversification, and minimum variance: An analytic perspective. *Journal of Portfolio Management.*

SRL Global Ltd. (2024, September 16). Backtest portfolio asset allocation. *Portfolio Visualizer.* Retrieved September 16, 2024, from https://www.portfoliovisualizer.com/backtest-Portfolio?s=y&sl=5M8RhWX2CyedGtfIDJRlyP

Project GitHub Repository: https://github.com/littlecl42/AAA500_FinalProject_CL_IL/

Appendix A

Projected Project Timeline/Deliverables

Week 3	9/16/2024	Form Teams, Choose Project/Dataset	Ian/Carrie
	9/21/2024	In-Person Meeting, Create Collab Files/Folder	Ian/Carrie
	9/22/2024	Formulate Research Questions	Ian
	9/23/2024	Generate Business Objective	Carrie
	9/23/2024	Review Dataset, Generate Data Descriptives	Carrie
Week 4	9/24/2024	Outline Data Analysis Plan	Ian
	9/28/2024	Develop Preliminary Models	Ian/Carrie
	9/28/2024	In-Person Meeting, Analyze Models	Ian/Carrie
	9/29/2024	Develop Conclusions and Recommendations	Ian/Carrie
	9/30/2024	Submit Final Project Check-in	Ian
Week 5	10/1/2024	Draft Report	Ian
	10/1/2024	Draft Presentation	Carrie
	10/5/2024	In-Person Meeting, Review Drafts	Ian/Carrie
Week 6	10/8/2024	Review Tables/Figures/Visualizations	Ian/Carrie
	10/8/2024	Review References/Sources/Links	Ian/Carrie
	10/11/2024	Finalize Final Report	Carrie
	10/12/2024	In-Person Meeting Presentation Rehearse	Ian/Carrie
	10/12/2024	Finalize Presentation	Ian
	10/12/2024	Record Project Presentation	Ian/Carrie
Week 7	10/21/2024	Submit Final Report /Presentation	Ian

Appendix B

Jupyter Notebook/Code

Appendix B1 - Ian Exploratory Data Analysis

Appendix B2 - Carrie Final Project Data Analysis

Appendix B1

Ian Exploratory Data Analysis 2024-10-17

October 18, 2024

1 Load and view a portion of the data (not an exhibit)

[2]:
```
import pandas as pd

# Load the dataset to inspect the contents
file_path = 'Opportunity_Set.xlsx'
data = pd.read_excel(file_path)

# Display the first few rows of the dataset to understand its structure
data.head()
```

[2]:

	Date	Vanguard LifeStrategy Income Fund (VASIX)	Vanguard Total World Stock ETF (VT)	PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)	AQR Diversified Arbitrage I (ADAIX)	iShares Gold Trust (IAU)
0	2014-11-30	0.0094	0.0126	0.0414	-0.0066	-0.0053
1	2014-12-31	-0.0005	-0.0199	0.0612	-0.0117	0.0133
2	2015-01-31	0.0141	-0.0163	0.1600	-0.0059	0.0865
3	2015-02-28	0.0033	0.0595	-0.1007	0.0069	-0.0579
4	2015-03-31	0.0018	-0.0121	0.0117		

```
4                                0.0000                    -0.0222

   Bitcoin Market Price USD (^BTC)  \
0                           0.0969
1                          -0.1777
2                          -0.2677
3                           0.1062
4                          -0.0150

   AQR Risk-Balanced Commodities Strategy I (ARCIX)  \
0                                           -0.0726
1                                           -0.0412
2                                           -0.0287
3                                            0.0044
4                                           -0.0573

   AQR Long-Short Equity I (QLEIX)  AQR Style Premia Alternative I (QSPIX)  \
0                           0.0248                                  0.0412
1                           0.0140                                  0.0002
2                           0.0156                                 -0.0112
3                           0.0236                                 -0.0390
4                          -0.0027                                  0.0256

   AQR Equity Market Neutral I (QMNIX)  AQR Macro Opportunities I (QGMIX)  \
0                               0.0257                               0.0154
1                               0.0195                               0.0039
2                               0.0290                              -0.0070
3                              -0.0078                               0.0091
4                               0.0049                               0.0261

   AGF U.S. Market Neutral Anti-Beta (BTAL)  \
0                                    0.0235
1                                    0.0294
2                                    0.0320
3                                   -0.0568
4                                    0.0000

   AQR Managed Futures Strategy HV I (QMHIX)  \
0                                     0.1159
1                                     0.0461
2                                     0.0721
3                                    -0.0108
4                                     0.0655

   Invesco DB US Dollar Bullish (UUP)  ProShares VIX Mid-Term Futures (VIXM)
0                              0.0165                                -0.0298
1                              0.0213                                 0.0553
```

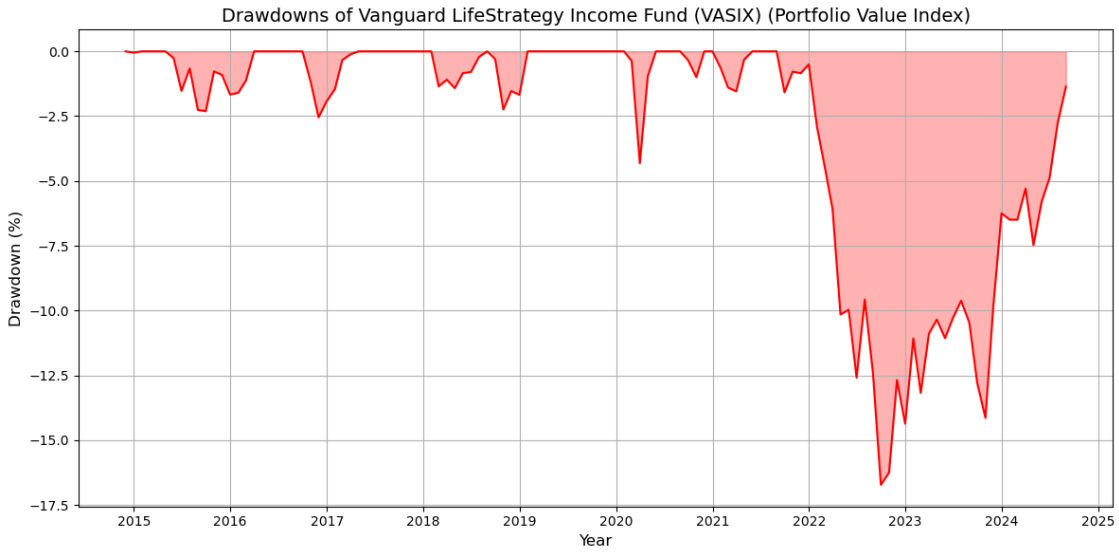
2	0.0484	0.0762
3	0.0028	-0.1145
4	0.0278	0.0033

2 Benchmark Asset Drawdown Analysis (Figure 1)

[2]:
```
import os
import matplotlib.pyplot as plt
import pandas as pd

# ------------------------------ #
#         Load the Data          #
# ------------------------------ #

# Define the file path
file_path = 'Opportunity_Set.xlsx'  # Adjust the path if necessary

# Load the dataset
data = pd.read_excel(file_path)

# Ensure 'Date' is set as the index
if 'Date' in data.columns:
    data = data.set_index('Date')

# Sort the index to ensure chronological order
data = data.sort_index()

# ------------------------------ #
#   Calculate Portfolio Value    #
# ------------------------------ #

# Define the benchmark asset
benchmark_column = "Vanguard LifeStrategy Income Fund (VASIX)"

# Check if the benchmark exists in the data
if benchmark_column not in data.columns:
    raise ValueError(f"Benchmark '{benchmark_column}' not found in the dataset␣
 ↪columns.")

# Extract the monthly returns for VASIX
benchmark_returns = data[benchmark_column].dropna()

# Set an initial investment amount
initial_value = 10000  # Starting portfolio value
```

```
# Create the portfolio value index by applying cumulative growth based on␣
 ↪returns
portfolio_value = initial_value * (1 + benchmark_returns).cumprod()

# ------------------------------ #
#       Calculate Drawdowns      #
# ------------------------------ #

# Calculate the cumulative maximum portfolio value
cumulative_max = portfolio_value.cummax()

# Calculate the drawdown as the percentage difference from the cumulative max
drawdowns = (portfolio_value - cumulative_max) / cumulative_max

# ------------------------------ #
#     Prepare Output Folder      #
# ------------------------------ #

# Define output folder and filename
output_folder = "VASIX_Drawdown_Plots"
if not os.path.exists(output_folder):
    os.makedirs(output_folder)  # Create the folder if it doesn't exist

# Define the filename for the plot
filename = os.path.join(output_folder, "VASIX_Drawdowns_Portfolio_Value.png")

# ------------------------------ #
#     Plot the Drawdowns         #
# ------------------------------ #

plt.figure(figsize=(12, 6))
plt.plot(drawdowns.index, drawdowns * 100, color='red', label='Drawdown',␣
 ↪linewidth=1.5)  # In percentage terms
plt.fill_between(drawdowns.index, drawdowns * 100, 0, color='red', alpha=0.3)

# Adding labels and title
plt.xlabel("Year", fontsize=12)
plt.ylabel("Drawdown (%)", fontsize=12)
plt.title(f"Drawdowns of {benchmark_column} (Portfolio Value Index)",␣
 ↪fontsize=14)

# Customize tick sizes for clarity
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)

# Display grid
plt.grid(True)
```

```
plt.tight_layout()

# Save the plot to the active directory
plt.savefig(filename, bbox_inches='tight', dpi=300)

# Display the plot in the notebook
plt.show()
```

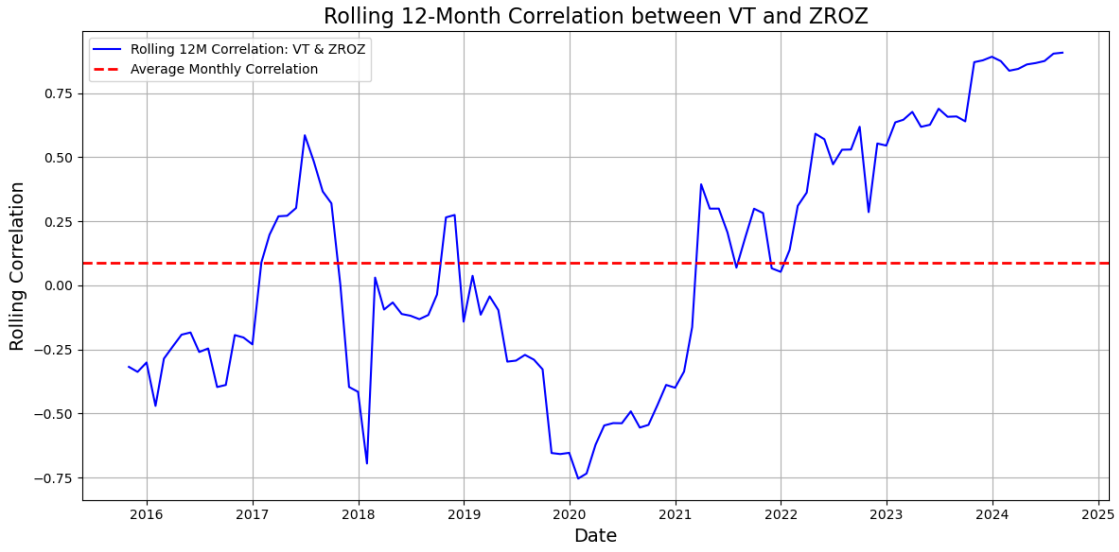


3 Rolling 12-month Correlation Between Stocks and Bonds (Figure 2)

[3]:
```
import os
import matplotlib.pyplot as plt
import pandas as pd

# ------------------------------ #
#         Load the Data          #
# ------------------------------ #

# Define the file path
file_path = 'Opportunity_Set.xlsx'

# Load the dataset
data = pd.read_excel(file_path)

# Ensure 'Date' is set as the index
if 'Date' in data.columns:
```

```
    data = data.set_index('Date')

# Sort the index to ensure chronological order
data = data.sort_index()

# ------------------------------ #
#     Prepare Output Folder      #
# ------------------------------ #

# Define a descriptive output folder name
output_folder = "Rolling_12M_Correlation_VT_ZROZ"
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# ------------------------------ #
#   Calculate Rolling Correlation #
# ------------------------------ #

# Define rolling window size
window_size = 12  # 12 months

# Extract returns for VT and ZROZ
vt_returns = data['Vanguard Total World Stock ETF (VT)'].dropna()
zroz_returns = data['PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)'].dropna()

# Align returns and calculate rolling 12-month correlation
aligned_returns = pd.concat([vt_returns, zroz_returns], axis=1).dropna()
rolling_corr = aligned_returns['Vanguard Total World Stock ETF (VT)'].
 ↪rolling(window=window_size).corr(aligned_returns['PIMCO 25+ Year Zero Coupon␣
 ↪US Trs ETF (ZROZ)'])

# ------------------------------ #
#    Define Plotting Function    #
# ------------------------------ #

def plot_rolling_correlation(corr_series, asset1, asset2, output_folder):
    """
    Plots rolling 12-month correlation between two assets.

    Parameters:
    - corr_series: pd.Series, the rolling correlation series
    - asset1: str, name of the first asset
    - asset2: str, name of the second asset
    - output_folder: str, path to the folder where the plot will be saved
    """
    plt.figure(figsize=(12, 6))
```

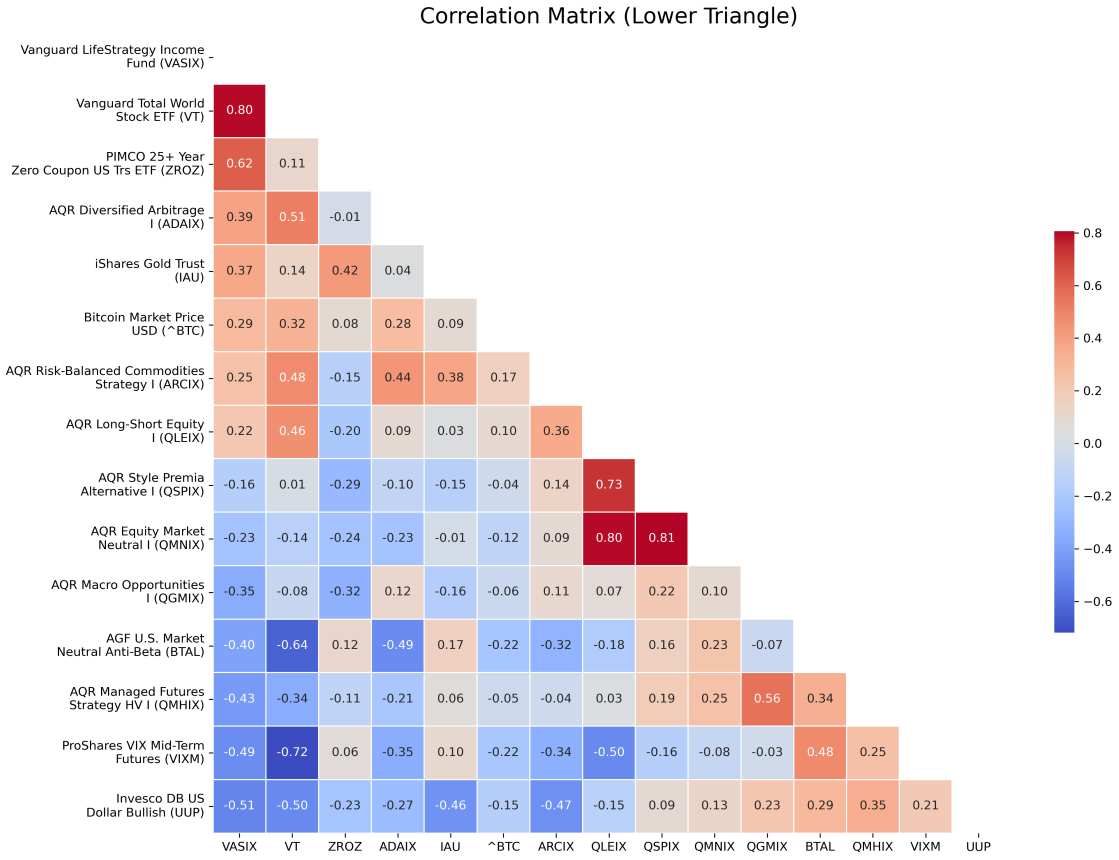
```
    plt.plot(corr_series.index, corr_series, color='blue', label=f'Rolling 12M␣
↪Correlation: {asset1} & {asset2}')

    avg_correlation = corr_series.mean()
    plt.axhline(y=avg_correlation, color='red', linestyle='--', linewidth=2,␣
↪label='Average Monthly Correlation')

    plt.xlabel("Date", fontsize=14)
    plt.ylabel("Rolling Correlation", fontsize=14)
    plt.title(f"Rolling 12-Month Correlation between {asset1} and {asset2}",␣
↪fontsize=16)
    plt.legend(loc='upper left')
    plt.grid(True)
    plt.tight_layout()

    # Define and save the plot
    filename = f"{asset1}_vs_{asset2}_Rolling12M_Correlation.png"
    plt.savefig(os.path.join(output_folder, filename), bbox_inches='tight',␣
↪dpi=300)
    plt.show()

# ------------------------------ #
#   Plot the Rolling Correlation  #
# ------------------------------ #

# Plot the rolling correlation between VT and ZROZ
plot_rolling_correlation(rolling_corr, 'VT', 'ZROZ', output_folder)
```



4 Correlation Matrix Heatmap (Lower Triangle) (Figure 3)

[4]:
```
import os
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
from PIL import Image
from IPython.display import display

# Define the file path
file_path = 'Opportunity_Set.xlsx'

# Load the dataset
data = pd.read_excel(file_path)

# Ensure 'Date' is set as the index
if 'Date' in data.columns:
    data = data.set_index('Date')

# Sort the index to ensure chronological order
data = data.sort_index()

# Sort the assets based on their correlation with VASIX
benchmark_column = "Vanguard LifeStrategy Income Fund (VASIX)"
correlations_with_vasix = data.corr()[benchmark_column].
 ↪sort_values(ascending=False)
data = data[correlations_with_vasix.index]

# Extract tickers from the column names (text between parentheses) and reorder␣
 ↪them based on sorted data
tickers = [data.columns.str.extract(r'\((.*?)\)')[0][i] for i in range(len(data.
 ↪columns))]

# ------------------------------ #
#   Generate High-Resolution Correlation Table #
# ------------------------------ #

def save_correlation_matrix_plot(data, tickers, output_folder):
    """
    Generates and saves a high-resolution PNG of the correlation matrix for all␣
 ↪assets.

    Parameters:
    - data: pd.DataFrame, the data for which to calculate the correlation matrix
    - tickers: list of str, the ticker symbols to use as labels
    - output_folder: str, path to the folder where the plot will be saved
```

```
    """
    # Calculate the correlation matrix
    corr_matrix = data.corr()

    # Set up the matplotlib figure
    plt.figure(figsize=(14, 10))

    # Draw the heatmap
    mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
    sns.heatmap(corr_matrix, mask=mask, annot=True, cmap='coolwarm',␣
 ↪linewidths=0.5, cbar_kws={"shrink": 0.5}, fmt=".2f")

    plt.title("Correlation Matrix (Lower Triangle)", fontsize=18)

    # Use the extracted tickers for xticks, ensuring the order matches the␣
 ↪sorted columns
    plt.xticks(ticks=np.arange(len(tickers)) + 0.5, labels=tickers,␣
 ↪fontsize=10, rotation=0)

    # Use the full asset names for yticks, with multi-line formatting if needed
    plt.yticks(ticks=np.arange(len(data.columns)) + 0.5, labels=[' '.join(label.
 ↪split()[:3]) + '\n' + ' '.join(label.split()[3:]) if len(label.split()) > 3␣
 ↪else label for label in data.columns], fontsize=10)

    plt.tight_layout()

    # Save the figure
    filename = "Correlation_Matrix_Lower_Triangle.png"
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)
    plt.savefig(os.path.join(output_folder, filename), bbox_inches='tight',␣
 ↪dpi=300)
    plt.close()

# Save the high-resolution correlation matrix plot
output_folder = "Correlation_Table"
save_correlation_matrix_plot(data, tickers, output_folder)

# Display the correlation matrix plot
corr_matrix_image_path = os.path.join(output_folder,␣
 ↪"Correlation_Matrix_Lower_Triangle.png")
display(Image.open(corr_matrix_image_path))
```



5 Rolling 12-month Average Asset Cross-Correlation (excluding VASIX) (Figure 4)

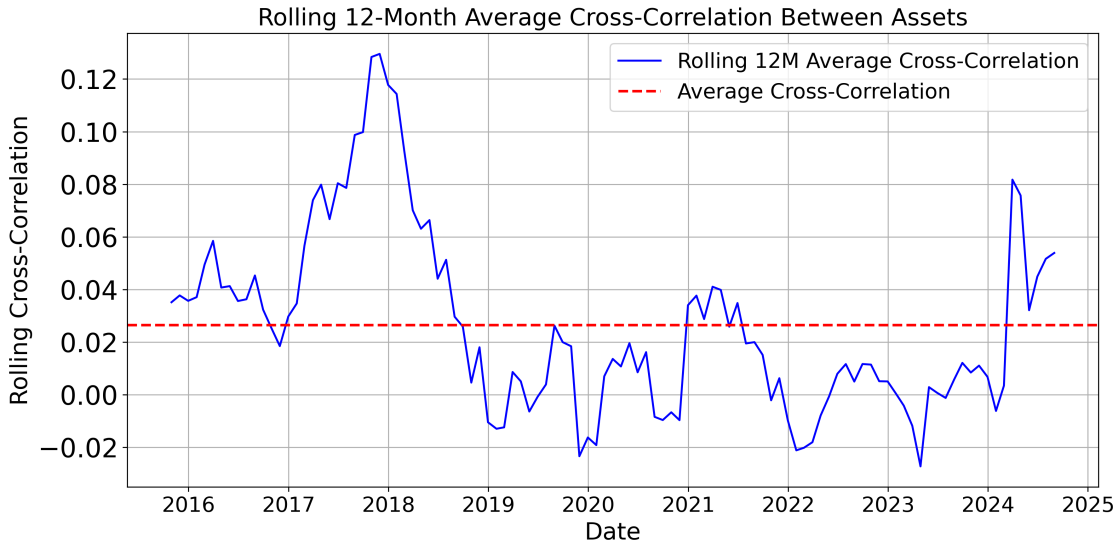
[6]:
```
import os
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from PIL import Image
from IPython.display import display
from scipy import stats
import numpy as np

# ------------------------------ #
#         Load the Data          #
# ------------------------------ #

# Define the file path
file_path = 'Opportunity_Set.xlsx'
```

```
# Load the dataset
data = pd.read_excel(file_path)

# Ensure 'Date' is set as the index
if 'Date' in data.columns:
    data = data.set_index('Date')

# Sort the index to ensure chronological order
data = data.sort_index()

# ------------------------------ #
#        Validate Benchmark      #
# ------------------------------ #

# Define the benchmark's full column name and its display label
benchmark_column = "Vanguard LifeStrategy Income Fund (VASIX)"
benchmark_label = 'VASIX'

# Check if the benchmark exists in the data
if benchmark_column not in data.columns:
    raise ValueError(f"Benchmark '{benchmark_label}' with column name␣
 ↪'{benchmark_column}' not found in the dataset columns.")

# Extract the data excluding the benchmark for rolling cross-correlation␣
 ↪calculations
assets_data = data.drop(columns=[benchmark_column]).dropna()

# Extract tickers from column names
tickers = assets_data.columns.str.extract(r'\((.*?)\)')[0].tolist()

# ------------------------------ #
#      Prepare Output Folder     #
# ------------------------------ #

# Define a descriptive output folder name
output_folder = "Rolling_12M_CrossCorrelation"

# Create the output folder if it doesn't exist
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# ------------------------------ #
#   Calculate Rolling Cross-Correlation #
# ------------------------------ #

# Define rolling window size
window_size = 12  # 12 months
```

```
# Get list of all asset columns
assets = assets_data.columns

# Initialize a DataFrame to store rolling cross-correlations
rolling_cross_correlations = pd.Series(index=assets_data.index, dtype=float)

# Loop through each time window to calculate cross-correlations
for start_idx in range(len(assets_data) - window_size + 1):
    window_end_idx = start_idx + window_size
    window_data = assets_data.iloc[start_idx:window_end_idx]

    # Calculate correlation matrix for the current window
    corr_matrix = window_data.corr()

    # Extract cross-correlation values (excluding 1.00 diagonal values)
    cross_corr_values = corr_matrix.values[np.triu_indices_from(corr_matrix,␣
 ↪k=1)]
    avg_cross_corr = np.nanmean(cross_corr_values)

    # Store the average cross-correlation value for the end of the window
    rolling_cross_correlations.iloc[window_end_idx - 1] = avg_cross_corr

# Drop NaN values from the rolling cross-correlation series
rolling_cross_correlations = rolling_cross_correlations.dropna()

# ------------------------------ #
#    Define Plotting Function    #
# ------------------------------ #

def plot_rolling_cross_correlation(cross_corr_series, output_folder):
    """
    Plots rolling 12-month average cross-correlation between all assets␣
 ↪excluding the benchmark.

    Parameters:
    - cross_corr_series: pd.Series, the rolling cross-correlation series
    - output_folder: str, path to the folder where the plot will be saved
    """
    plt.figure(figsize=(12, 6))
    plt.plot(cross_corr_series.index, cross_corr_series, color='blue',␣
 ↪label='Rolling 12M Average Cross-Correlation')

    # Calculate and plot the average cross-correlation over the full timespan
    avg_cross_corr = cross_corr_series.mean()
    plt.axhline(y=avg_cross_corr, color='red', linestyle='--', linewidth=2,␣
 ↪label='Average Cross-Correlation')
```

```
    plt.xlabel("Date", fontsize=18)  # Doubled the font size of x-axis label
    plt.ylabel("Rolling Cross-Correlation", fontsize=18)  # Doubled the font␣
 ↪size of y-axis label
    plt.xticks(fontsize=16)  # Doubled the font size of x-axis tick numbers
    plt.yticks(fontsize=20)  # Doubled the font size of y-axis tick numbers
    plt.title("Rolling 12-Month Average Cross-Correlation Between Assets",␣
 ↪fontsize=18)
    plt.legend(loc='upper right', fontsize=16)
    plt.grid(True)
    plt.tight_layout()

    # Define a clear and descriptive filename
    filename = "Rolling12M_Average_CrossCorrelation.png"
    plt.savefig(os.path.join(output_folder, filename), bbox_inches='tight',␣
 ↪dpi=300)
    plt.close()

# ------------------------------ #
#     Generate and Save Plot     #
# ------------------------------ #

# Plot and save the rolling cross-correlation
plot_rolling_cross_correlation(rolling_cross_correlations, output_folder)

# Display the plot
combined_image_path = os.path.join(output_folder,␣
 ↪"Rolling12M_Average_CrossCorrelation.png")
display(Image.open(combined_image_path))

# ------------------------------ #
#   Generate High-Resolution Correlation Table #
# ------------------------------ #

def save_correlation_matrix_plot(data, tickers, output_folder):
    """
    Generates and saves a high-resolution PNG of the correlation matrix␣
 ↪excluding the diagonal 1.00 values and upper right duplicates.

    Parameters:
    - data: pd.DataFrame, the data for which to calculate the correlation matrix
    - tickers: list of str, the ticker symbols to use as labels
    - output_folder: str, path to the folder where the plot will be saved
    """
    # Calculate the correlation matrix
    corr_matrix = data.corr()
```

```
    # Mask the upper triangle and diagonal
    mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
    np.fill_diagonal(mask, True)

    # Set up the matplotlib figure
    plt.figure(figsize=(14, 10))

    # Draw the heatmap with the mask
    sns.heatmap(corr_matrix, mask=mask, annot=True, cmap='coolwarm',␣
 ↪linewidths=0.5, cbar_kws={"shrink": 0.5}, fmt=".2f")

    plt.title("Correlation Matrix (Lower Triangle, Excluding Diagonal)",␣
 ↪fontsize=18)
    plt.xticks(ticks=np.arange(len(tickers)) + 0.5, labels=tickers,␣
 ↪fontsize=10, rotation=0)
    plt.yticks(ticks=np.arange(len(data.columns)) + 0.5, labels=[label.
 ↪replace(' (', '\n(') for label in data.columns], fontsize=11)
    plt.tight_layout()

    # Save the figure
    filename = "Correlation_Matrix_Lower_Triangle.png"
    plt.savefig(os.path.join(output_folder, filename), bbox_inches='tight',␣
 ↪dpi=300)
    plt.close()

# Save the high-resolution correlation matrix plot
save_correlation_matrix_plot(data, tickers, output_folder)

# Display the correlation matrix plot
#corr_matrix_image_path = os.path.join(output_folder,␣
 ↪"Correlation_Matrix_Lower_Triangle.png")
#display(Image.open(corr_matrix_image_path))
```



6 Asset Summary Statistics (Table 2)

[221]:
```
import pandas as pd
import dataframe_image as dfi

# Load the dataset to inspect the contents
file_path = 'Opportunity_Set.xlsx'
data = pd.read_excel(file_path)

# Drop the 'Date' column before running the summary statistics
numeric_data = data.drop(columns=['Date'])

# Calculate kurtosis and skew for each asset
kurtosis = numeric_data.kurtosis()
skewness = numeric_data.skew()

# Calculate the correlations of all assets with VASIX
correlation_with_vasix = numeric_data.corr()["Vanguard LifeStrategy Income Fund␣
 ↪(VASIX)"]

# Generate summary statistics for all assets
summary_stats = numeric_data.describe().T  # Transpose for readability

# Rename columns for better clarity
summary_stats.columns = ['Count', 'Mean', 'Std Dev', 'Min', '25%', 'Median',␣
 ↪'75%', 'Max']
```

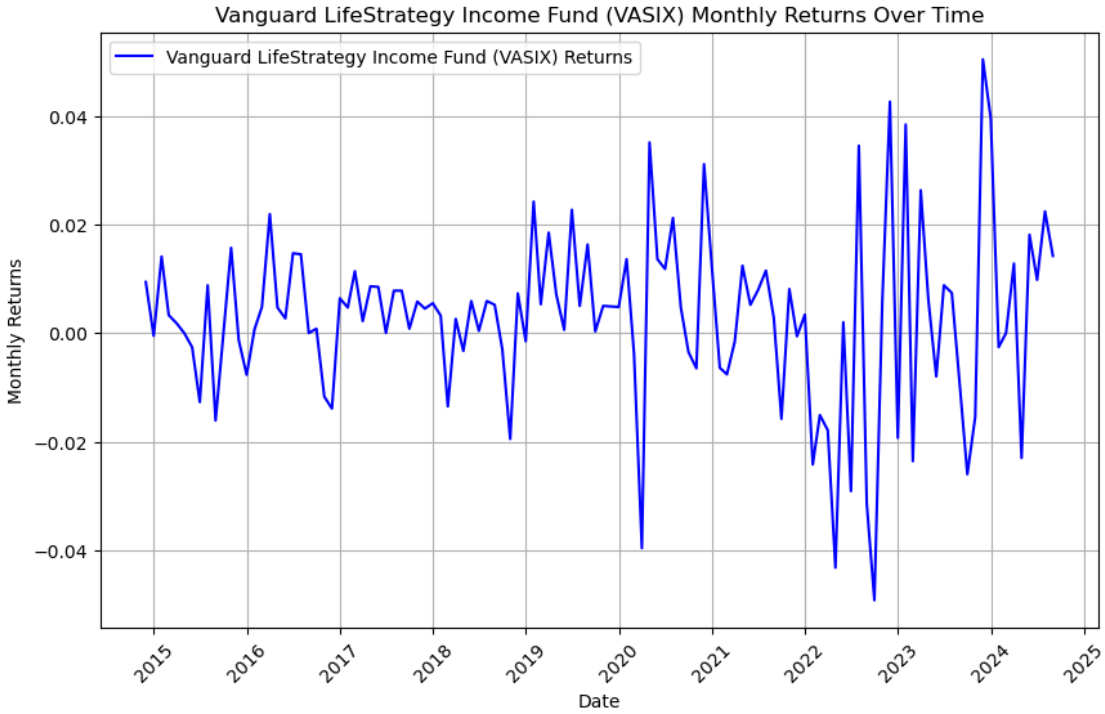
```
# Add Kurtosis and Skewness
summary_stats['Skewness'] = skewness
summary_stats['Kurtosis'] = kurtosis

# Convert count to integer
summary_stats['Count'] = summary_stats['Count'].astype(int)

# Convert only percentage-relevant columns (mean, std dev, etc.) to percentage␣
 ↪form
columns_to_convert = ['Mean', 'Std Dev', 'Min', '25%', 'Median', '75%', 'Max']
for col in columns_to_convert:
    summary_stats[col] = summary_stats[col] * 100  # Convert to percentage form

# Add the correlation with VASIX to the immediate right of the Count column
summary_stats.insert(1, 'Correlation with VASIX', correlation_with_vasix)

# Sort the summary stats by the correlation with VASIX
sorted_summary_stats = summary_stats.loc[correlation_with_vasix.
 ↪sort_values(ascending=False).index]

# Adjust the color gradient to make it warmer and show 2 decimal places for␣
 ↪skewness and kurtosis
styled_table = sorted_summary_stats.style \
    .format("{:.2f}", subset=pd.IndexSlice[:, columns_to_convert]) \
    .format("{:.0f}", subset=['Count']) \
    .format("{:.2f}", subset=['Correlation with VASIX', 'Skewness',␣
 ↪'Kurtosis']) \
    .set_caption("Summary Statistics of Asset Returns (in Percent Form,␣
 ↪Including Skewness and Kurtosis, Sorted by Correlation with VASIX)") \
    .set_table_styles([
        {'selector': 'caption', 'props': 'caption-side: top; font-size: 14px;␣
 ↪font-weight: bold; color: #6B6B6B;'},
        {'selector': 'thead th', 'props': [('background-color', '#f5f5f5'),␣
 ↪('color', 'black'), ('font-weight', 'bold')]}
    ]) \
    .background_gradient(cmap='Oranges', subset=['Correlation with VASIX'],␣
 ↪axis=0, low=0.2, high=0.8)  # Warm color gradient for correlation

# Save the styled table as an image (PNG format)
dfi.export(styled_table, 'formatted_table.png')

# Optional: Display the formatted table in Jupyter
styled_table
```

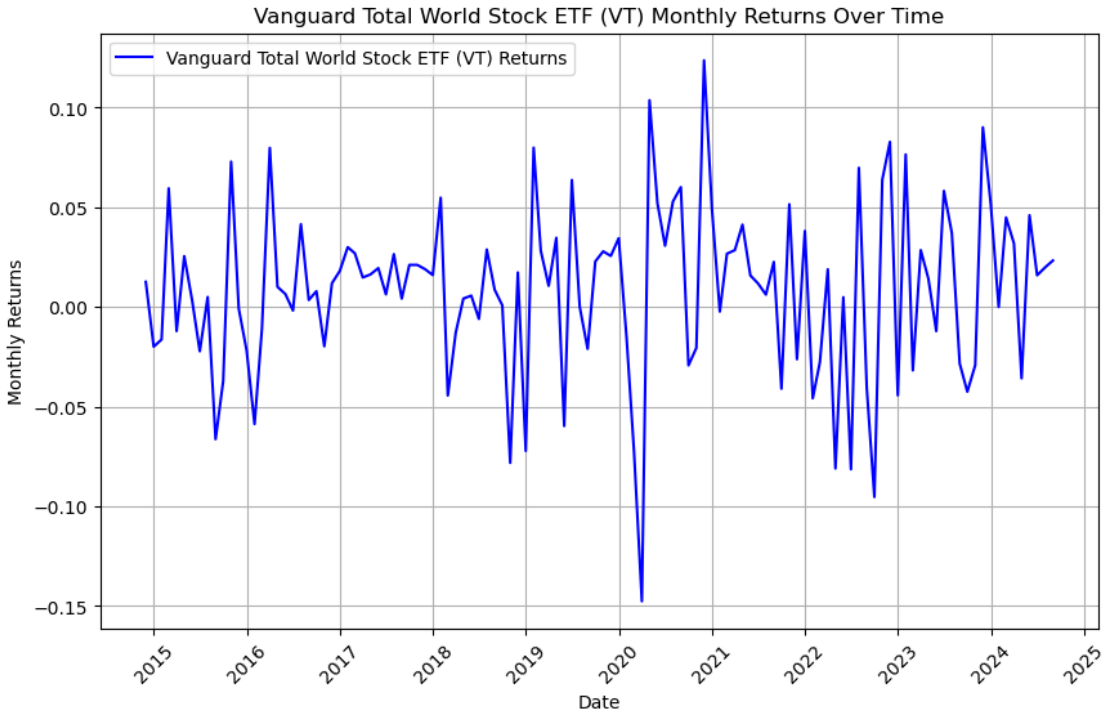
[221]: <pandas.io.formats.style.Styler at 0x30e4a9a30>

7 Monthly Return Time Series (Figure 5)

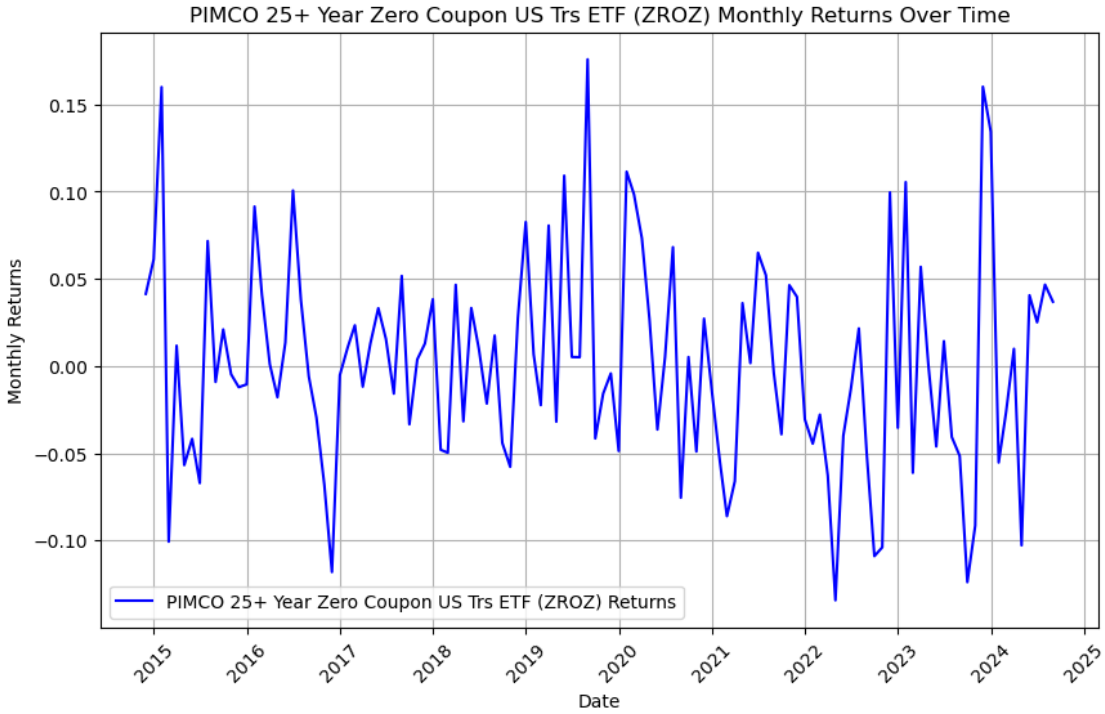
[11]:
```
import os
import matplotlib.pyplot as plt
from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
from matplotlib.figure import Figure
import numpy as np
import math  # Added to calculate rows and columns

# Comment-based snippet name for the folder
snippet_name = "MONTHLY_RETURN_PLOTS"

# Create the directory to save plots
output_folder = snippet_name
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# Create a list to hold individual figures
figures = []

# Generate time series plots for each asset (excluding the 'Date' index)
for asset in data.columns:  # Loop through all asset columns
    if asset != 'Date':  # Exclude 'Date'
        fig, ax = plt.subplots(figsize=(10, 6))
        ax.plot(data.index, data[asset], label=f"{asset} Returns", color='blue')
        ax.set_title(f"{asset} Monthly Returns Over Time")
        ax.set_xlabel("Date")
        ax.set_ylabel("Monthly Returns")
        ax.tick_params(axis='x', rotation=45)
        ax.grid(True)
        ax.legend()

        # Save the individual figure to the list
        figures.append(fig)

        # Save the figure as a PNG file inside the created folder
        fig.savefig(os.path.join(output_folder, f"{asset}_monthly_returns.
 ↪png"), bbox_inches='tight', dpi=300)  # Save with high resolution

        # Display the plot in Jupyter Notebook
        plt.show()

        # Close the figure to save memory
        plt.close(fig)

# Calculate number of rows and columns for the combined figure
num_figures = len(figures)
```

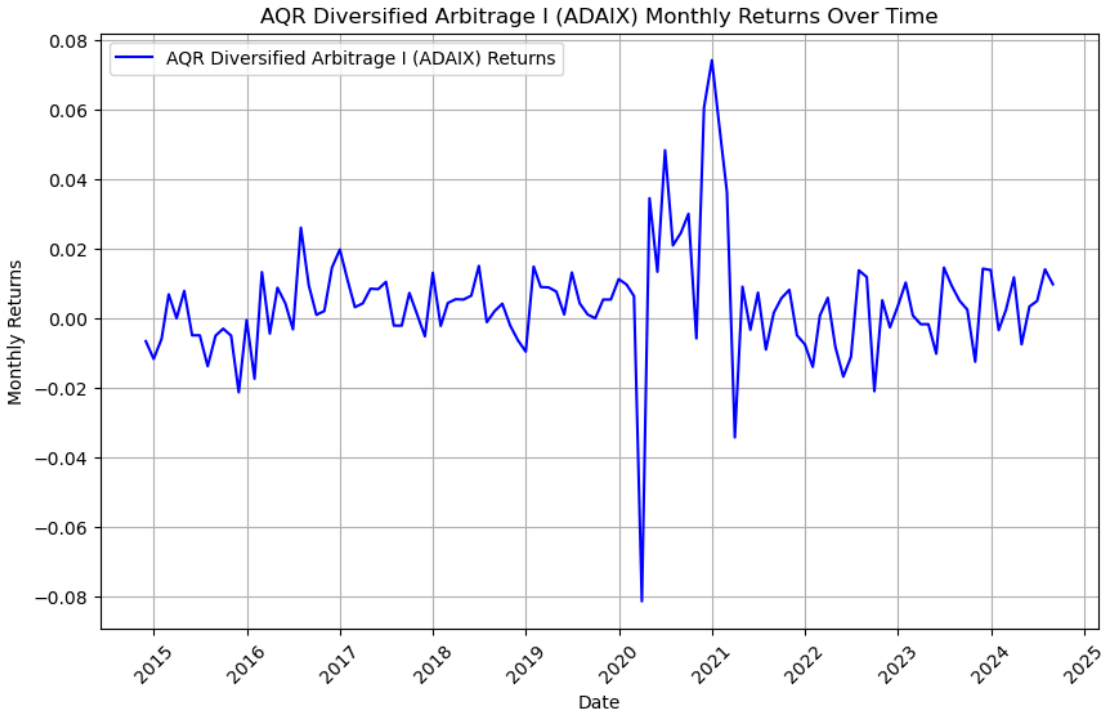
```
num_cols = 3  # You can adjust this to set the number of columns per row
num_rows = math.ceil(num_figures / num_cols)  # Calculate the necessary number␣
 ↪of rows

# Create a new figure to combine all plots into a single image
combined_fig, combined_axes = plt.subplots(num_rows, num_cols, figsize=(16,␣
 ↪22))  # Increase figure size for better readability
combined_axes = combined_axes.flatten()

# Add each individual figure to the combined figure
for i, fig in enumerate(figures):
    ax = combined_axes[i]
    for line in fig.axes[0].get_lines():
        ax.plot(line.get_xdata(), line.get_ydata(), label=line.get_label(),␣
 ↪color=line.get_color())
    ax.set_title(fig.axes[0].get_title(), fontsize=10)
    ax.set_xlabel(fig.axes[0].get_xlabel(), fontsize=8)
    ax.set_ylabel(fig.axes[0].get_ylabel(), fontsize=8)
    ax.tick_params(axis='x', rotation=45, labelsize=8)
    ax.tick_params(axis='y', labelsize=8)
    ax.grid(True)
    ax.legend(fontsize=8)

# Hide any unused subplots
for j in range(len(figures), len(combined_axes)):
    combined_fig.delaxes(combined_axes[j])

# Adjust layout to prevent overlap
combined_fig.tight_layout()

# Save the combined figure as a PNG file
combined_fig.savefig(os.path.join(output_folder, "combined_monthly_returns.
 ↪png"), bbox_inches='tight', dpi=300)  # Save with high resolution

# Display the combined plot in Jupyter Notebook
plt.show()
```



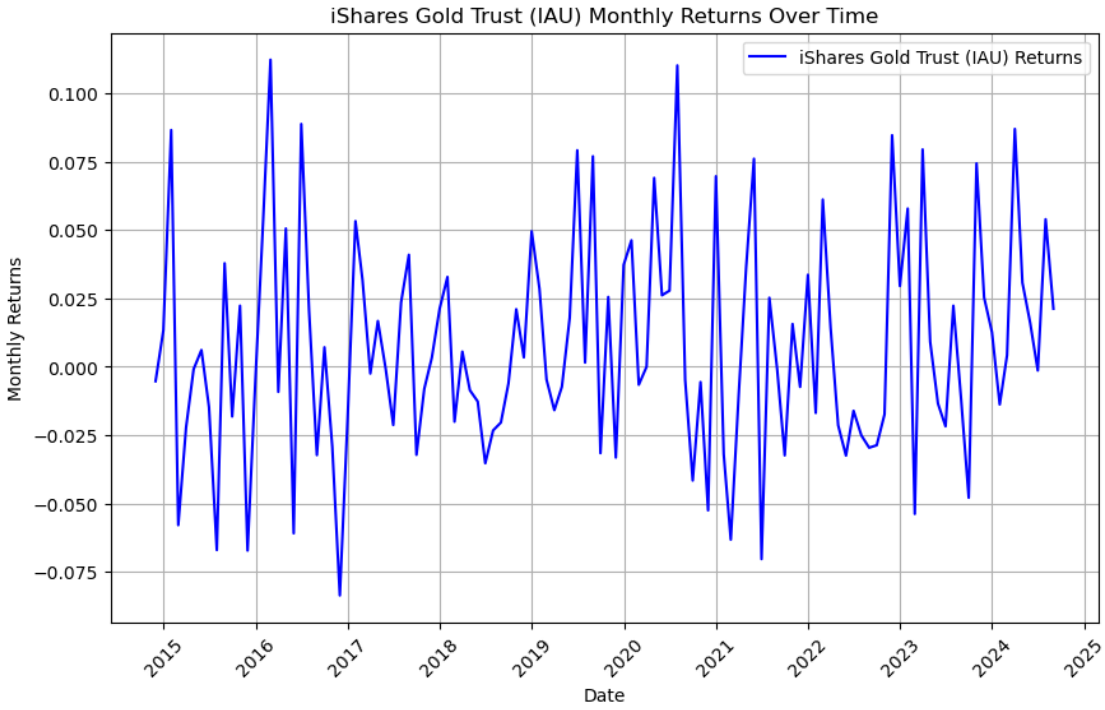


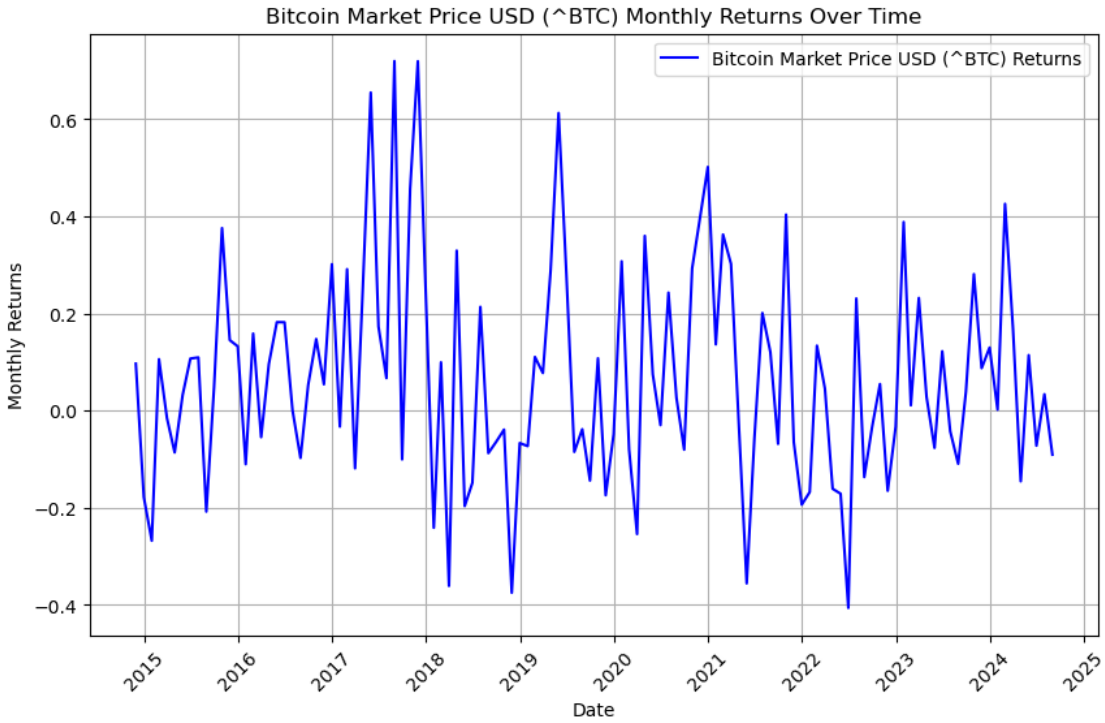
Appendix B1 - Ian Exploratory Data Analysis



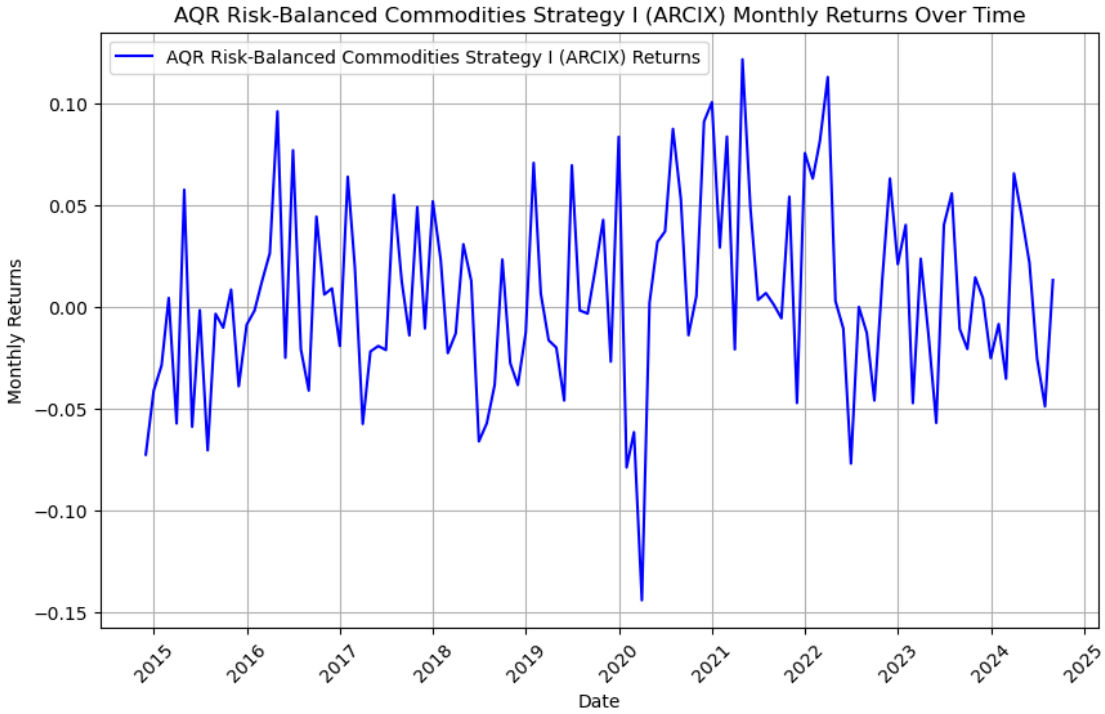


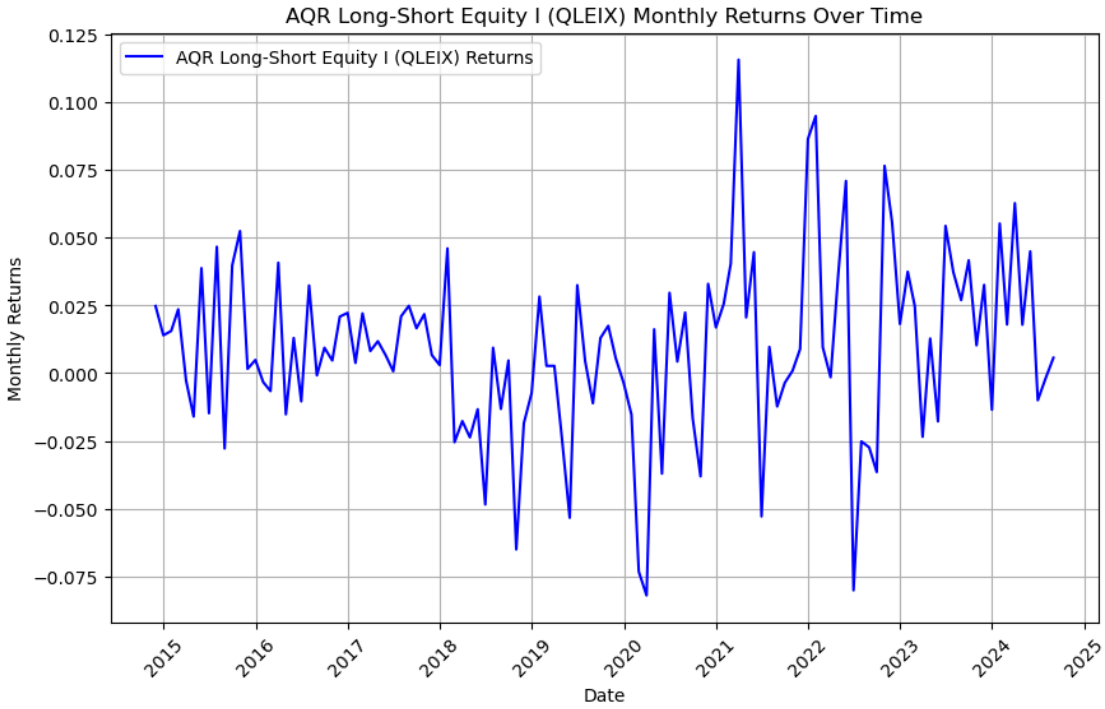
Appendix B1 - Ian Exploratory Data Analysis



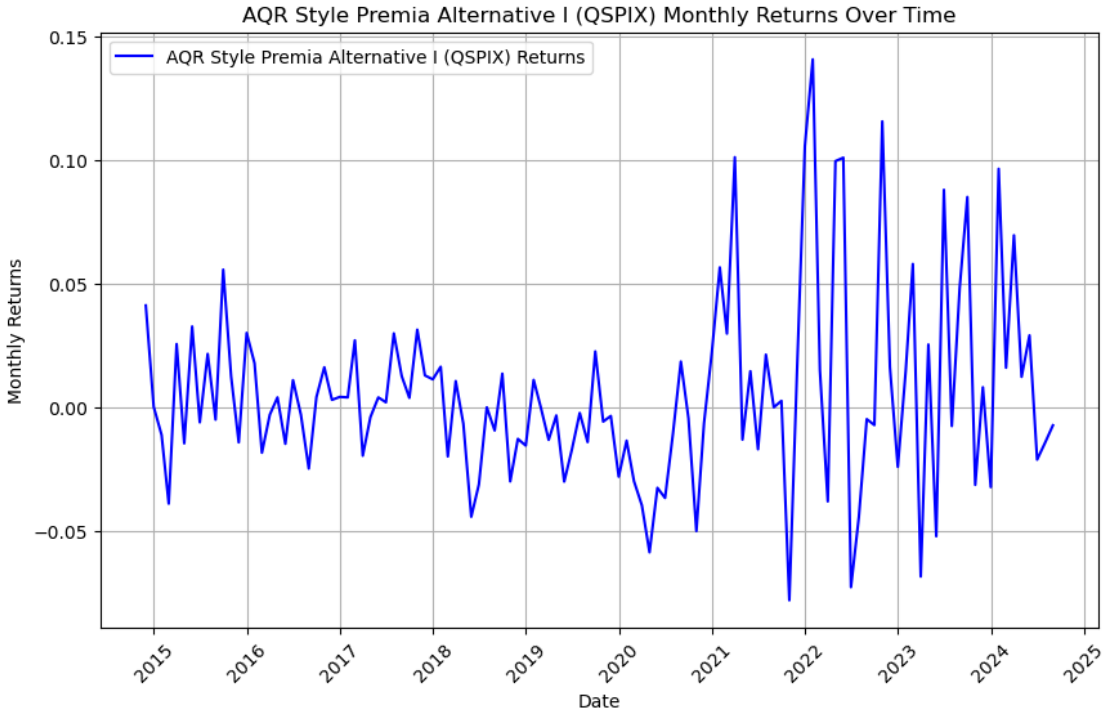


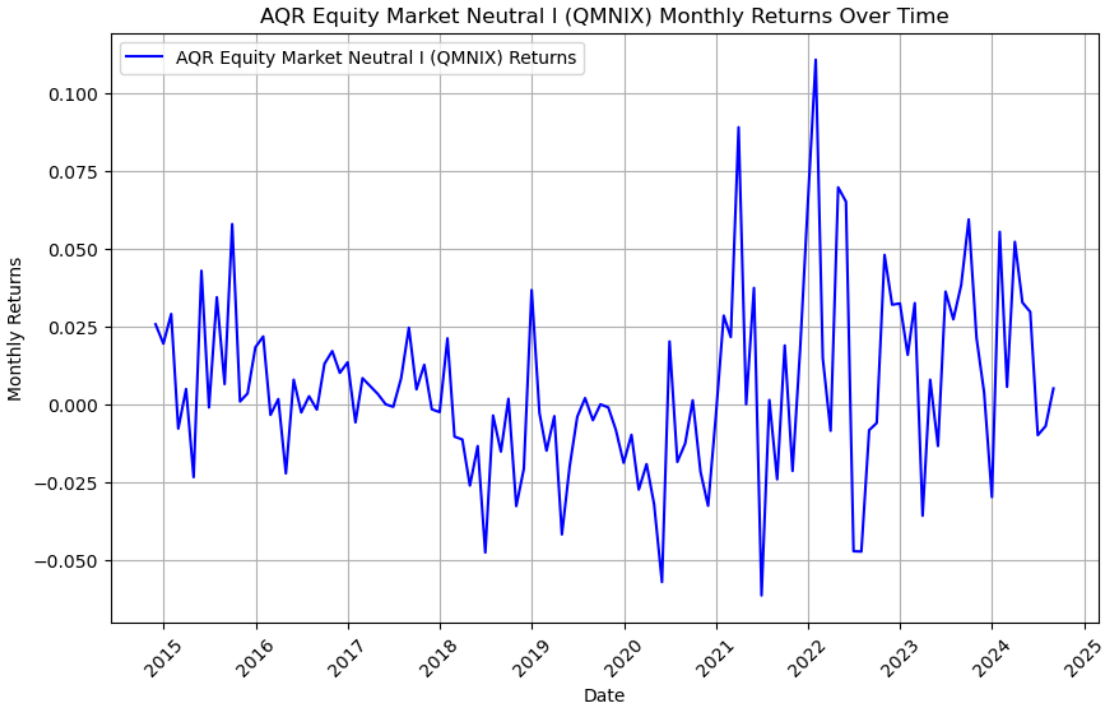
Appendix B1 - Ian Exploratory Data Analysis



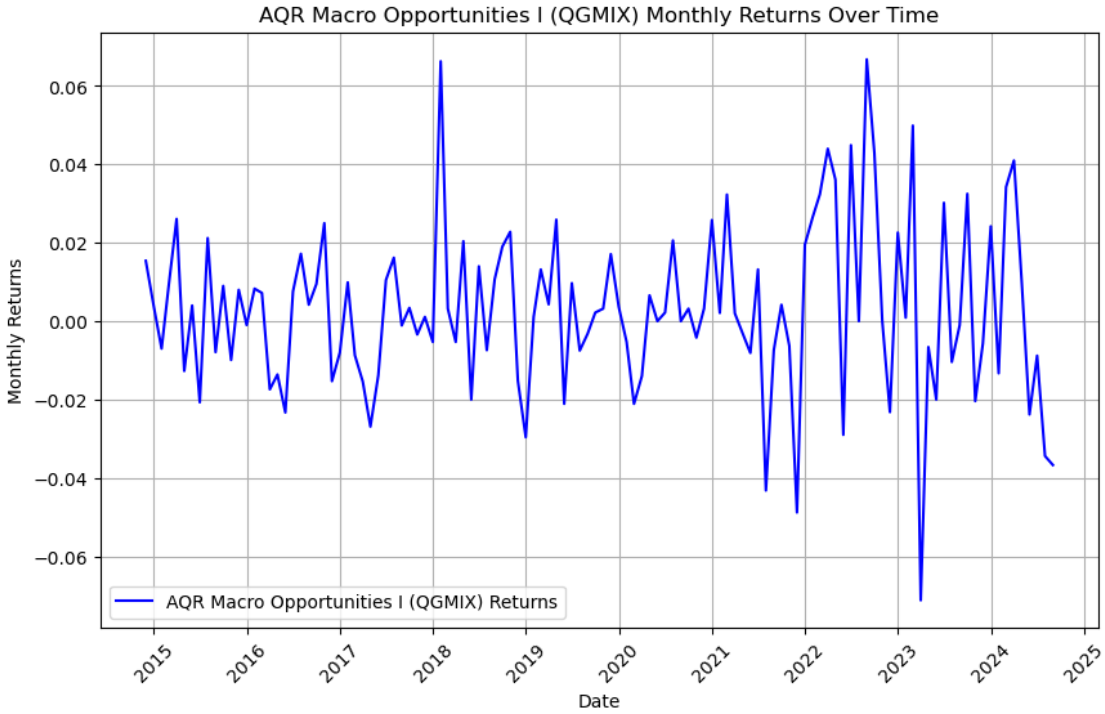


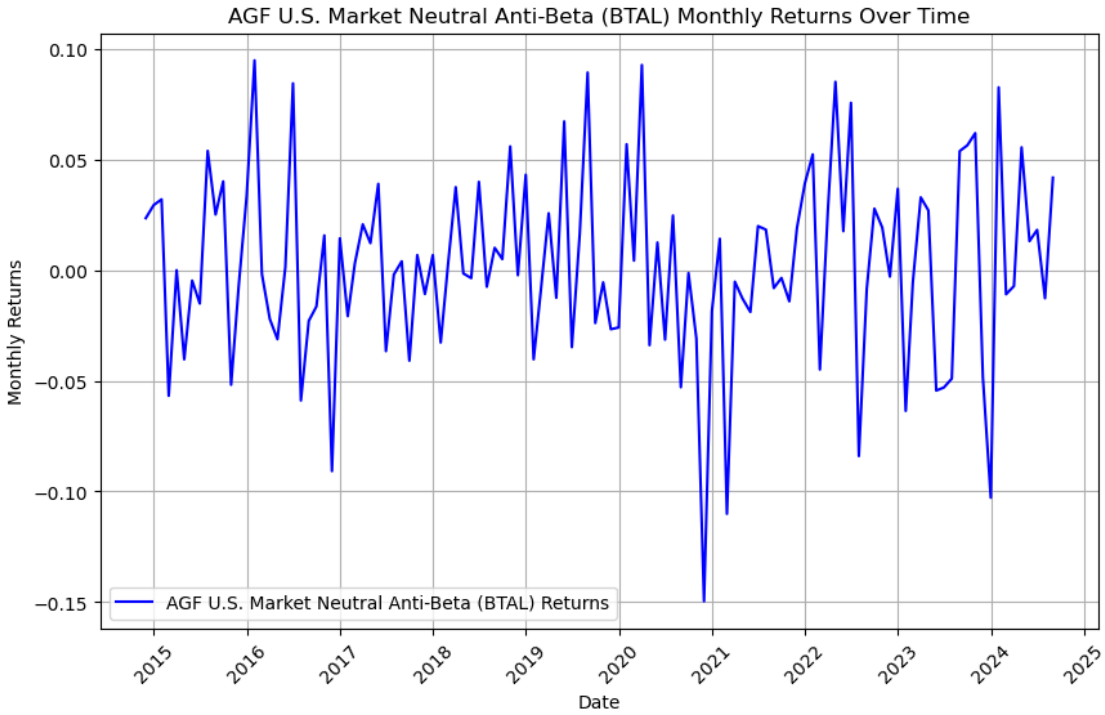
Appendix B1 - Ian Exploratory Data Analysis



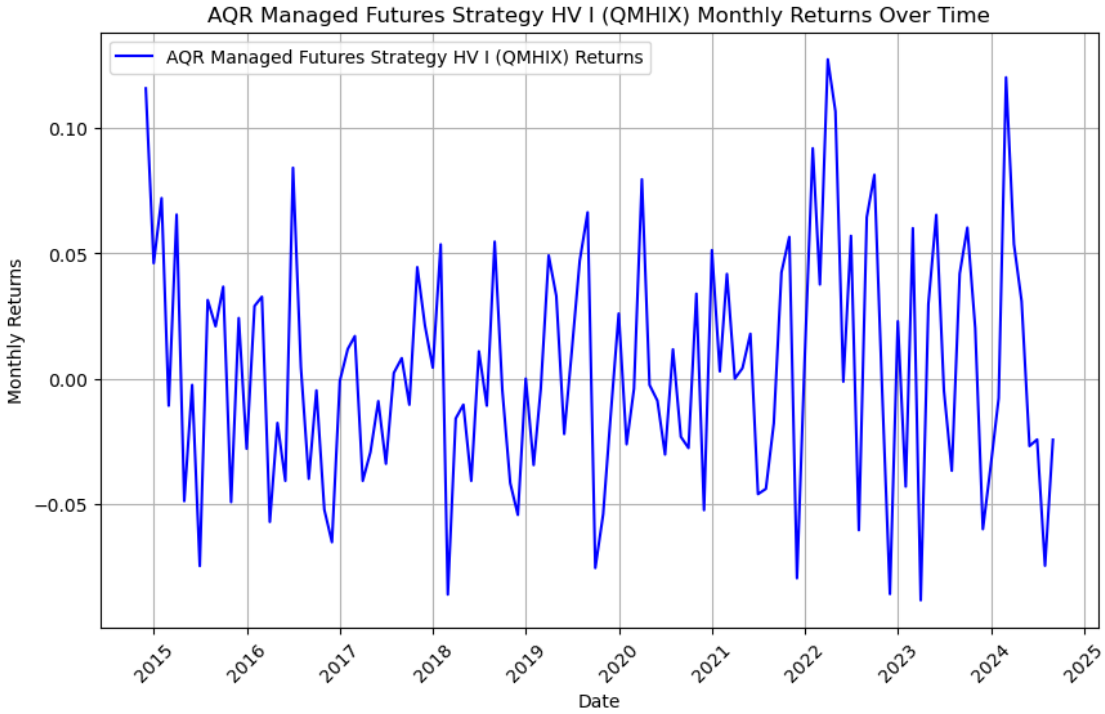


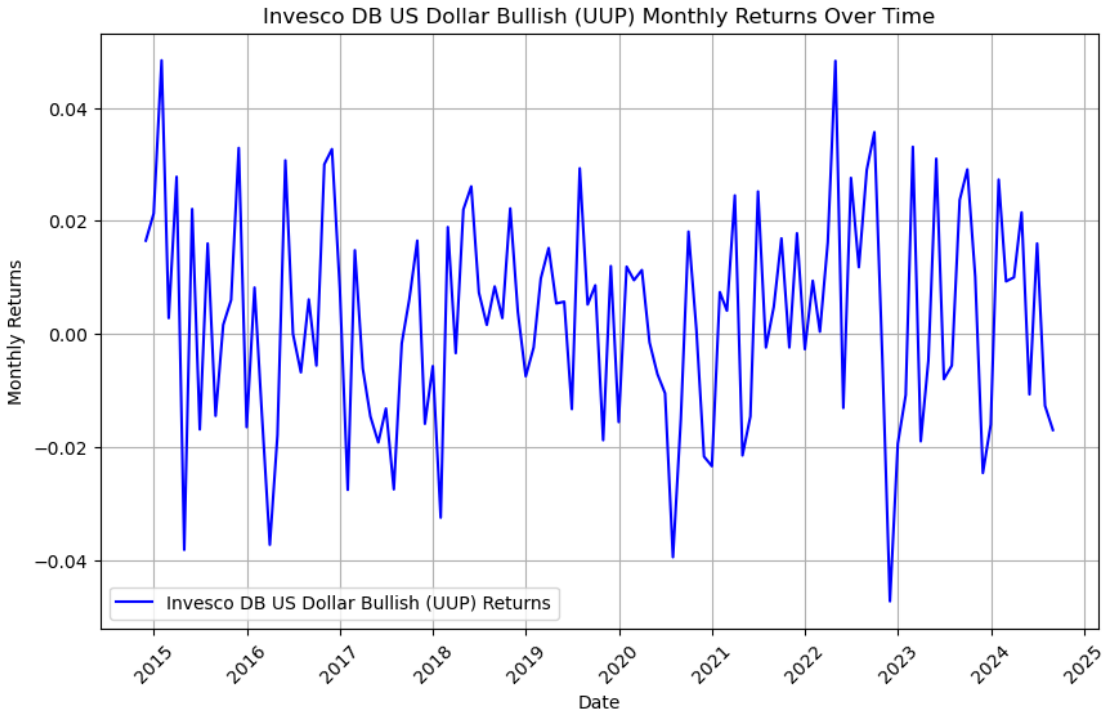
Appendix B1 - Ian Exploratory Data Analysis



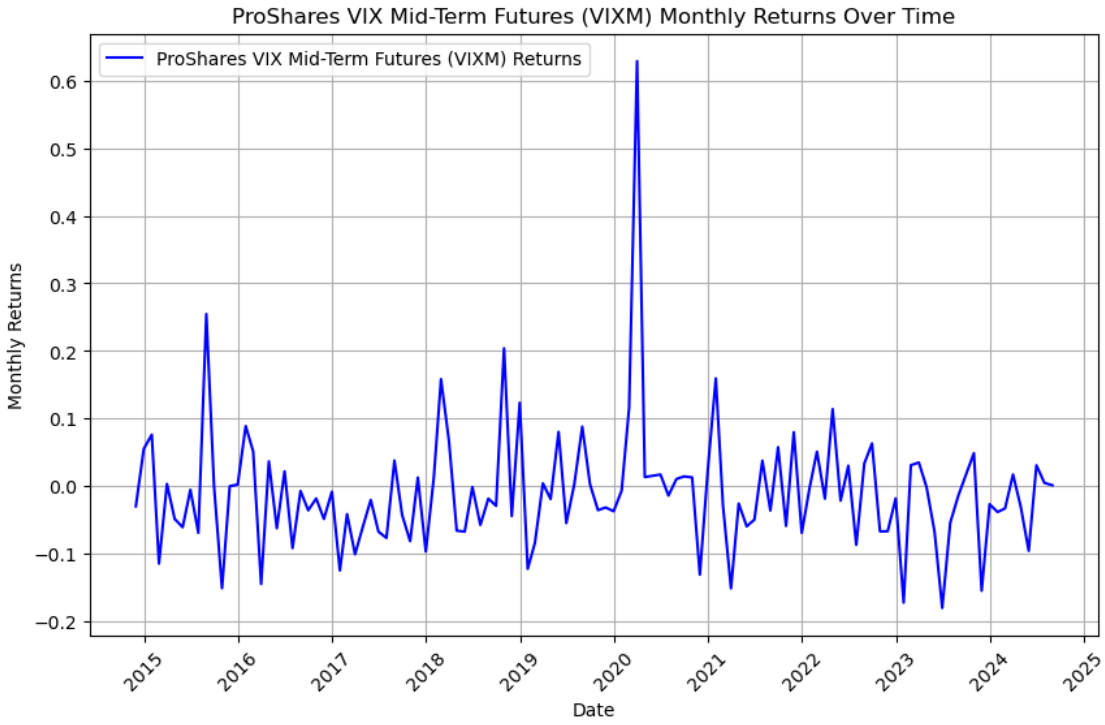


Appendix B1 - Ian Exploratory Data Analysis

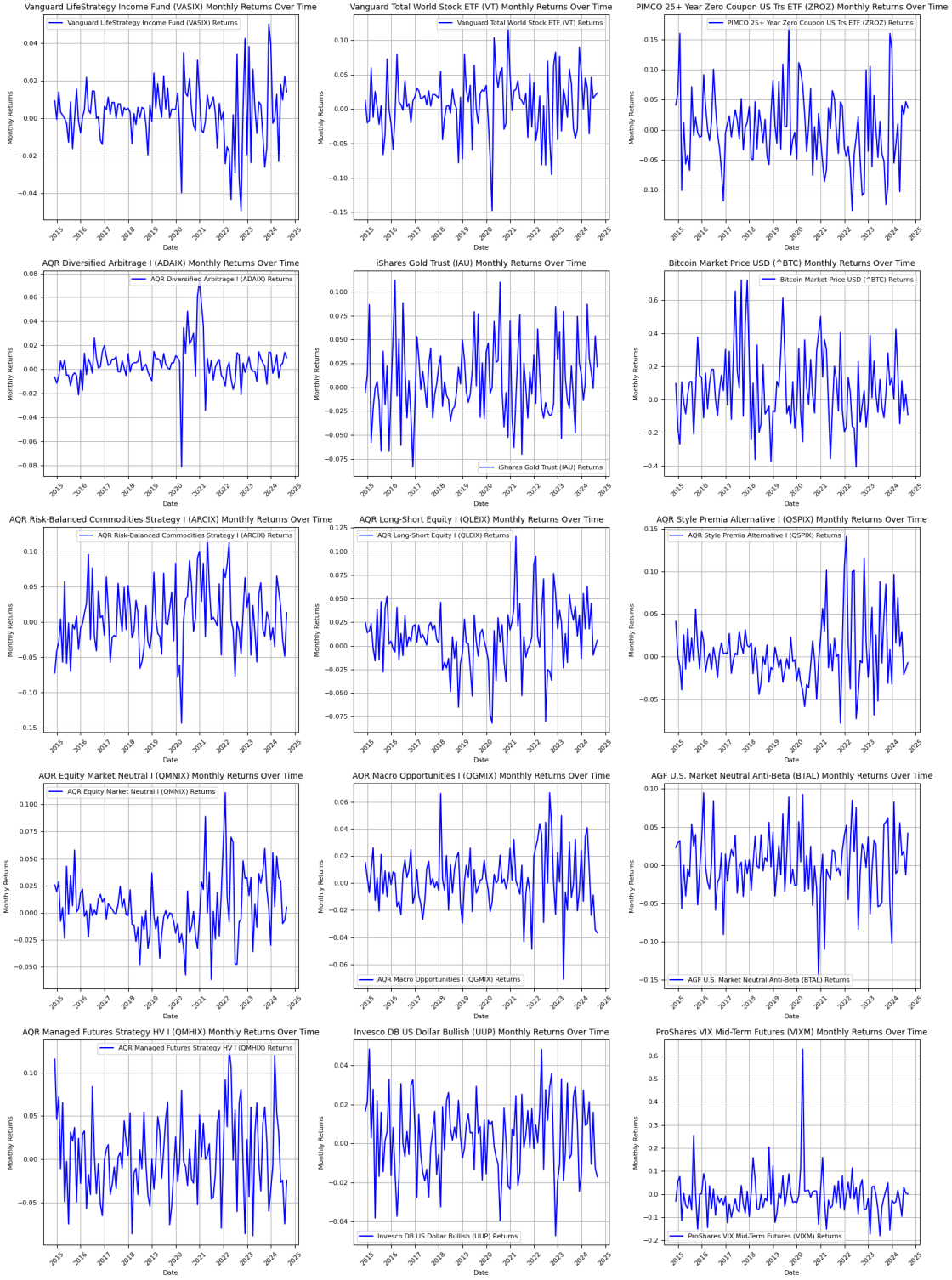




Appendix B1 - Ian Exploratory Data Analysis



Appendix B1 - Ian Exploratory Data Analysis

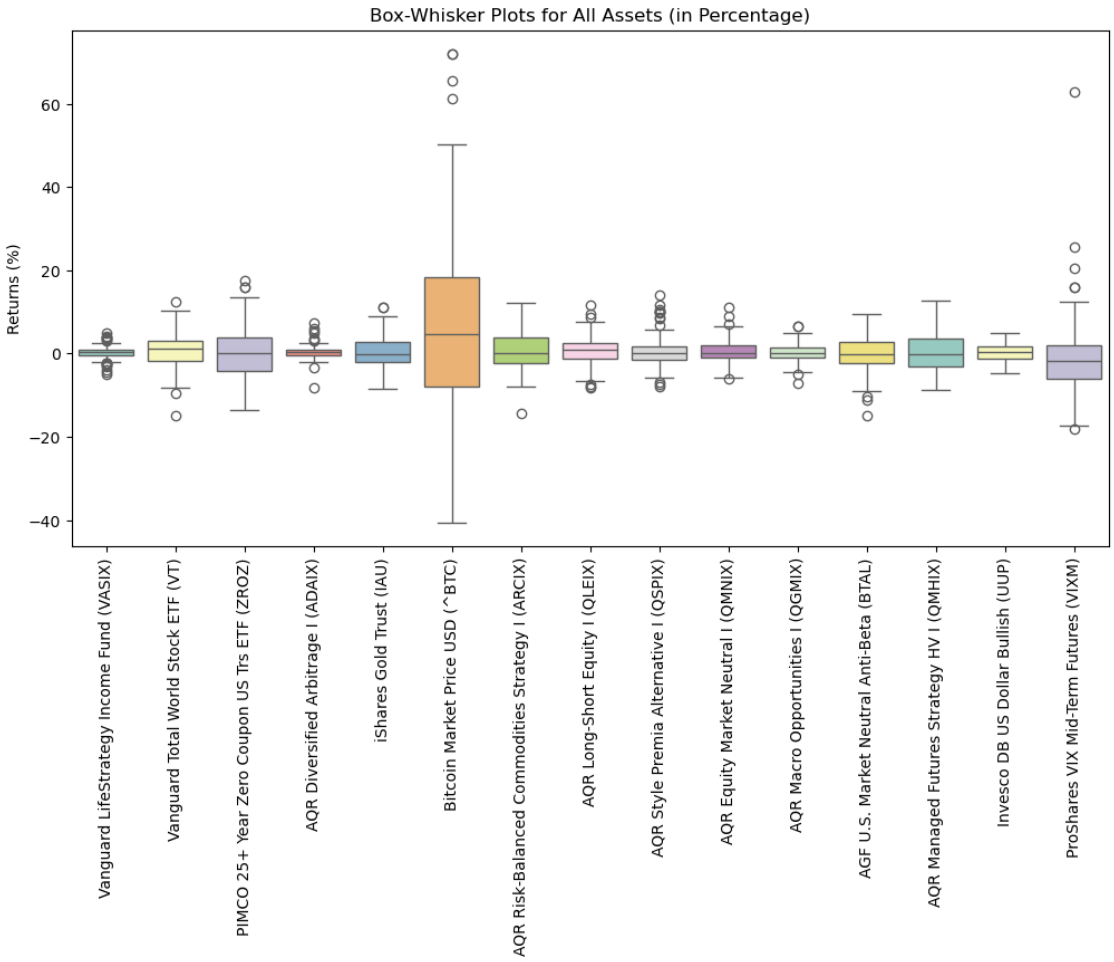


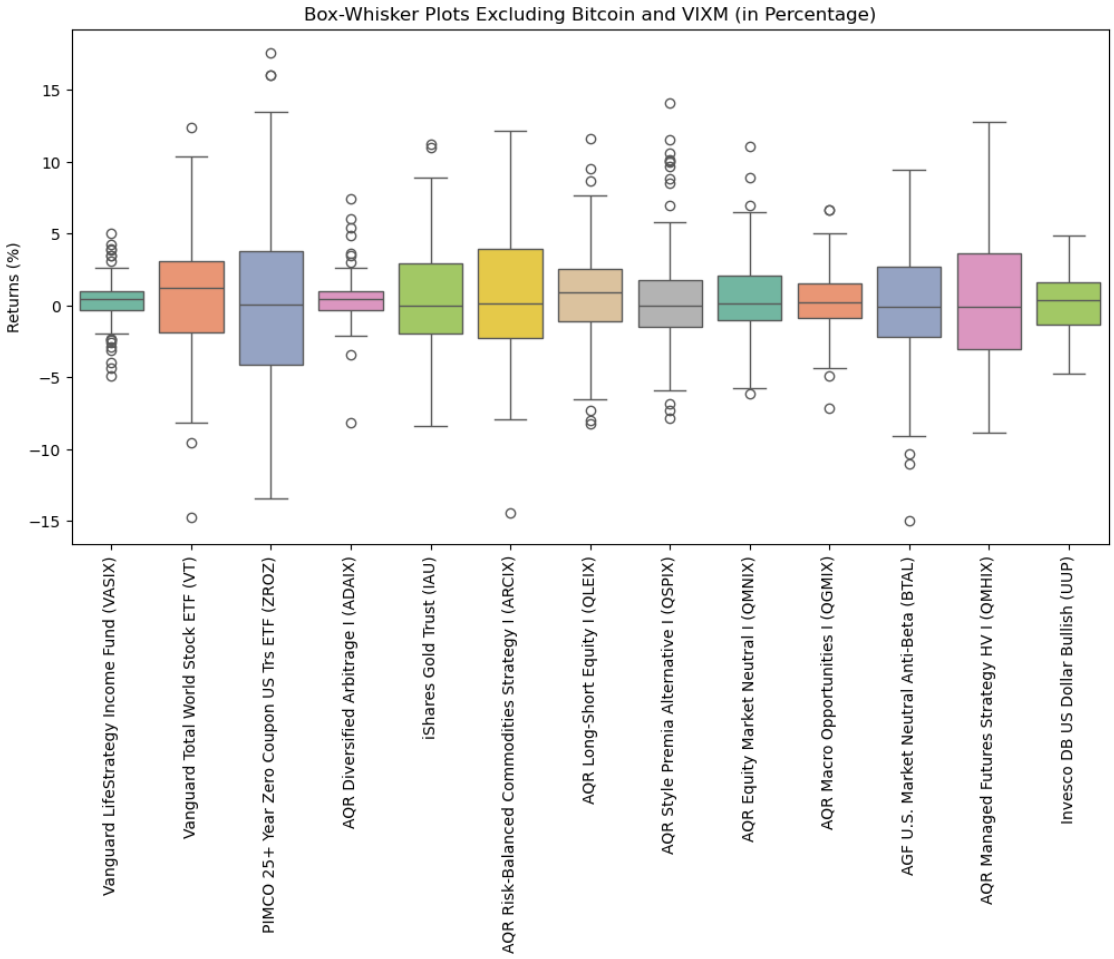
Appendix B1 - Ian Exploratory Data Analysis

8 Box-Whisker Plots (Figure 6)

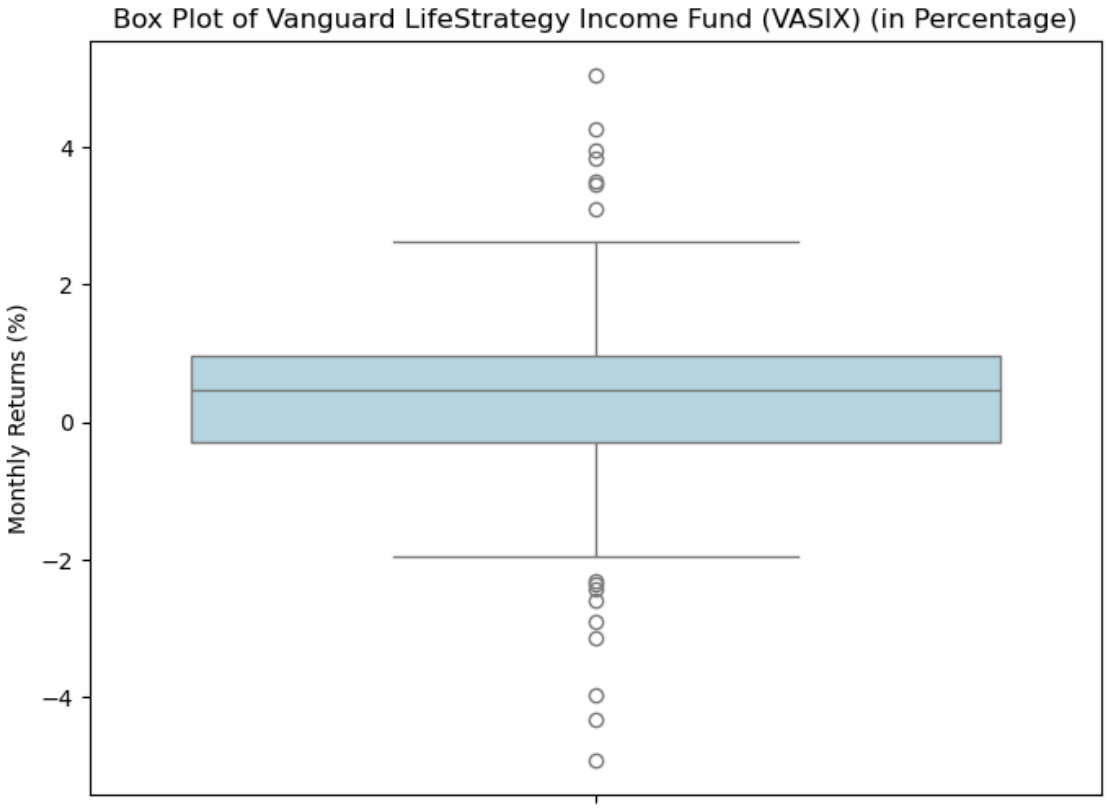
[12]:
```
#BOX AND WHISKER PLOTS

import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
file_path = 'Opportunity_Set.xlsx'
data = pd.read_excel(file_path)

# Multiply the data by 100 to convert to percentage form (excluding 'Date'␣
 ↪column)
data_percentage = data.drop(columns=['Date']) * 100

# Folder to store the plots
output_folder = "BOX_WHISKER_PLOTS"
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# Step 1: Generate box-whisker plots for all assets (percentages)
plt.figure(figsize=(12, 6))
sns.boxplot(data=data_percentage, palette="Set3")
plt.title("Box-Whisker Plots for All Assets (in Percentage)")
plt.ylabel("Returns (%)")
plt.xticks(rotation=90)
plt.savefig(os.path.join(output_folder, "box_whisker_all_assets.png"),␣
 ↪bbox_inches='tight', dpi=300)
plt.show()

# Step 2: Generate box-whisker plots excluding Bitcoin and VIXM
data_without_bitcoin_vixm = data_percentage.drop(columns=["Bitcoin Market Price␣
 ↪USD (^BTC)", "ProShares VIX Mid-Term Futures (VIXM)"])
plt.figure(figsize=(12, 6))
sns.boxplot(data=data_without_bitcoin_vixm, palette="Set2")
plt.title("Box-Whisker Plots Excluding Bitcoin and VIXM (in Percentage)")
plt.ylabel("Returns (%)")
plt.xticks(rotation=90)
plt.savefig(os.path.join(output_folder, "box_whisker_excluding_bitcoin_vixm.
 ↪png"), bbox_inches='tight', dpi=300)
plt.show()

# Step 3: Generate individual box-whisker plots for each asset (percentages)
for asset in data_percentage.columns:
    plt.figure(figsize=(8, 6))
```

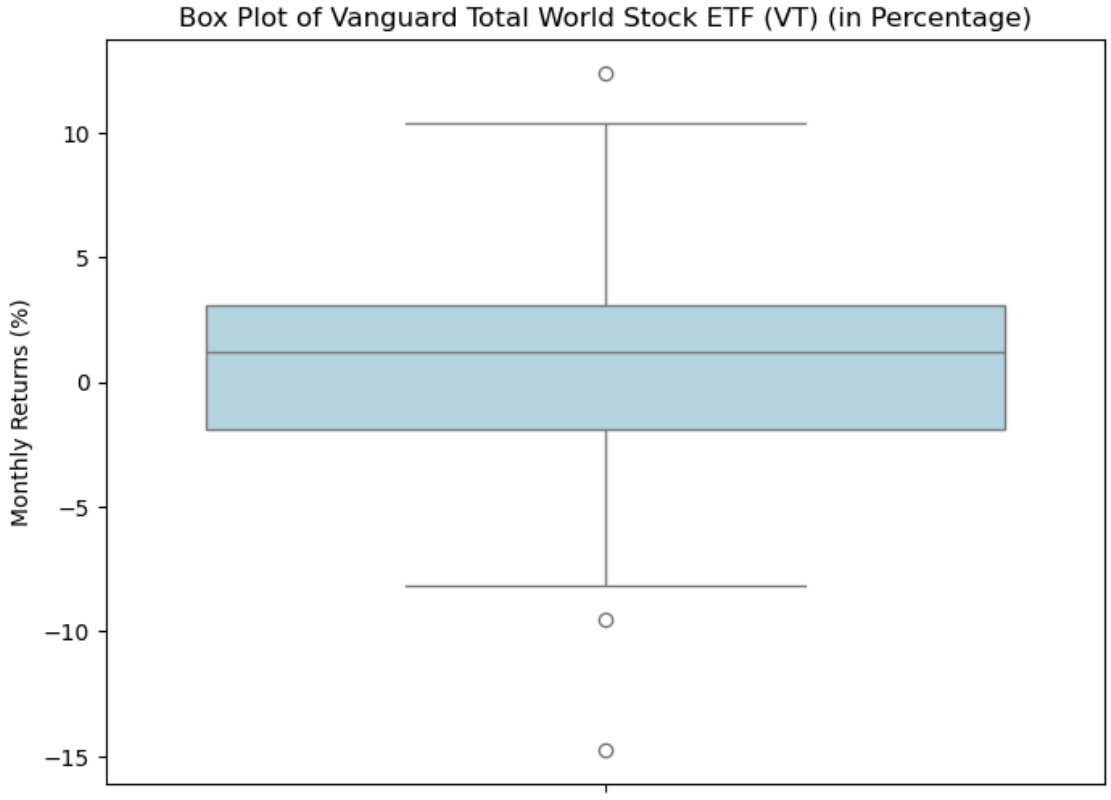
```
    sns.boxplot(data=data_percentage[asset], color='lightblue')
    plt.title(f"Box Plot of {asset} (in Percentage)")
    #plt.xlabel("Monthly Returns (%)")
    plt.ylabel("Monthly Returns (%)")
    plt.savefig(os.path.join(output_folder, f"{asset}_box_whisker.png"),␣
↪bbox_inches='tight', dpi=300)

    # Display the plot in Jupyter notebook
    plt.show()
```



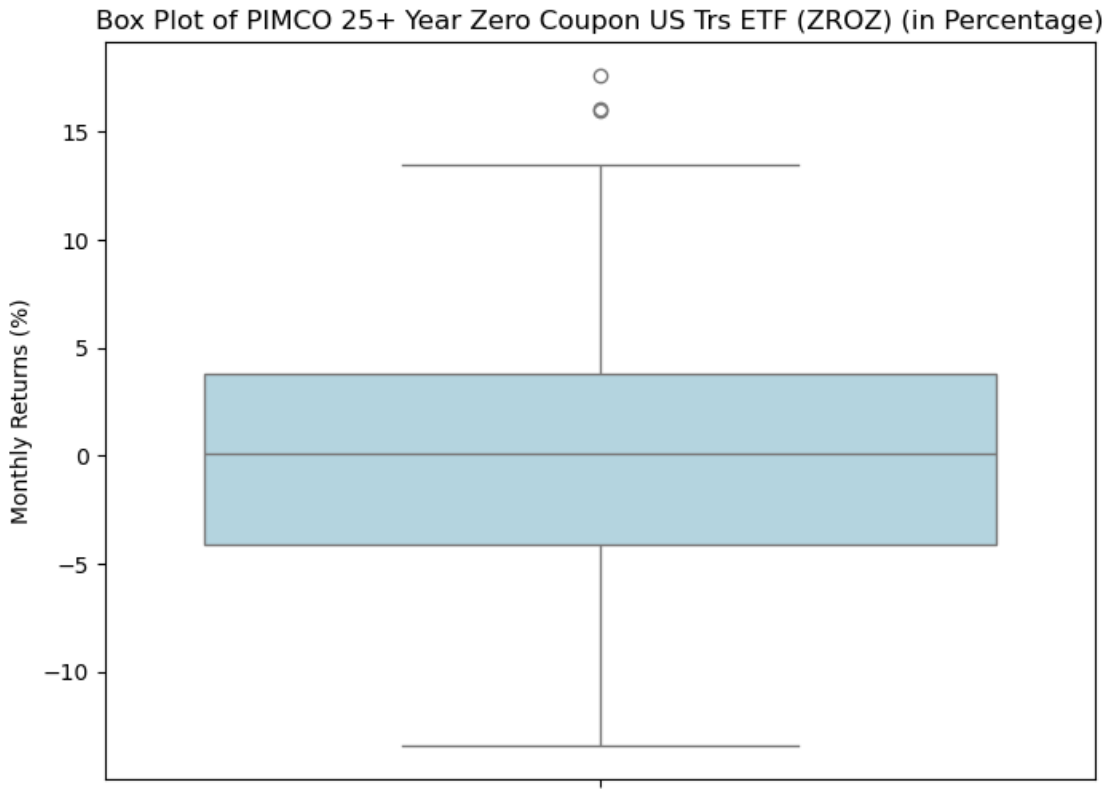


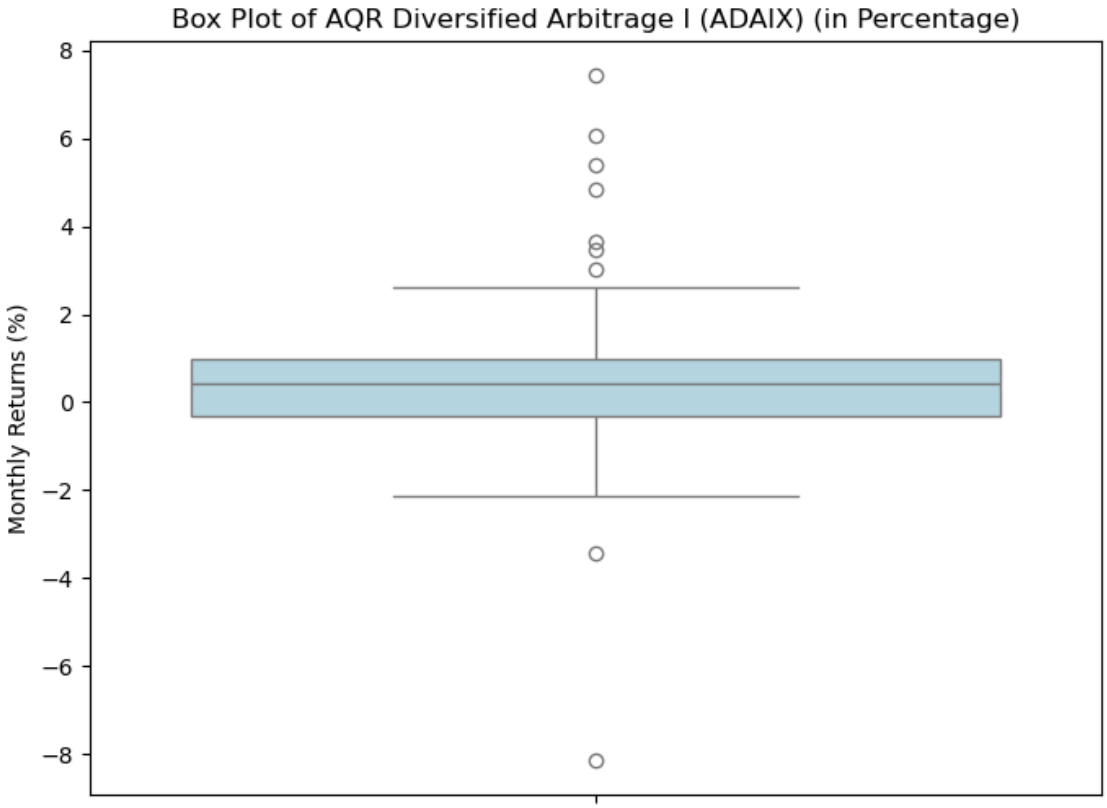
Appendix B1 - Ian Exploratory Data Analysis



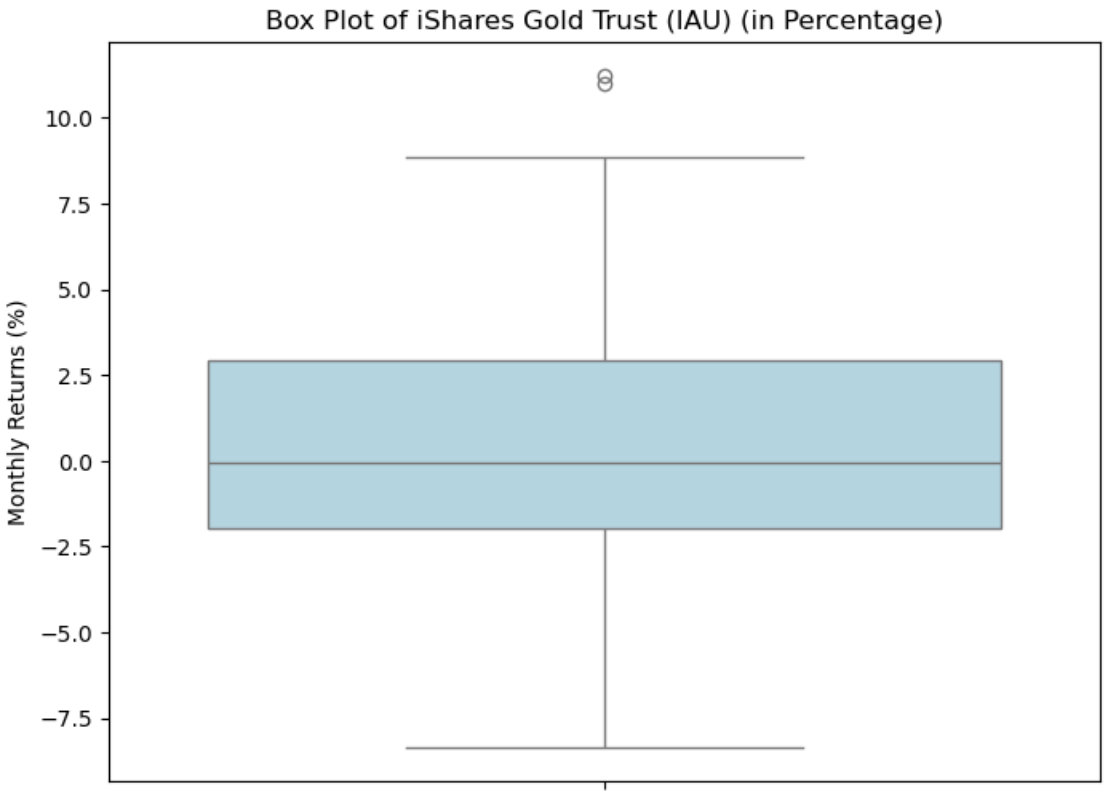


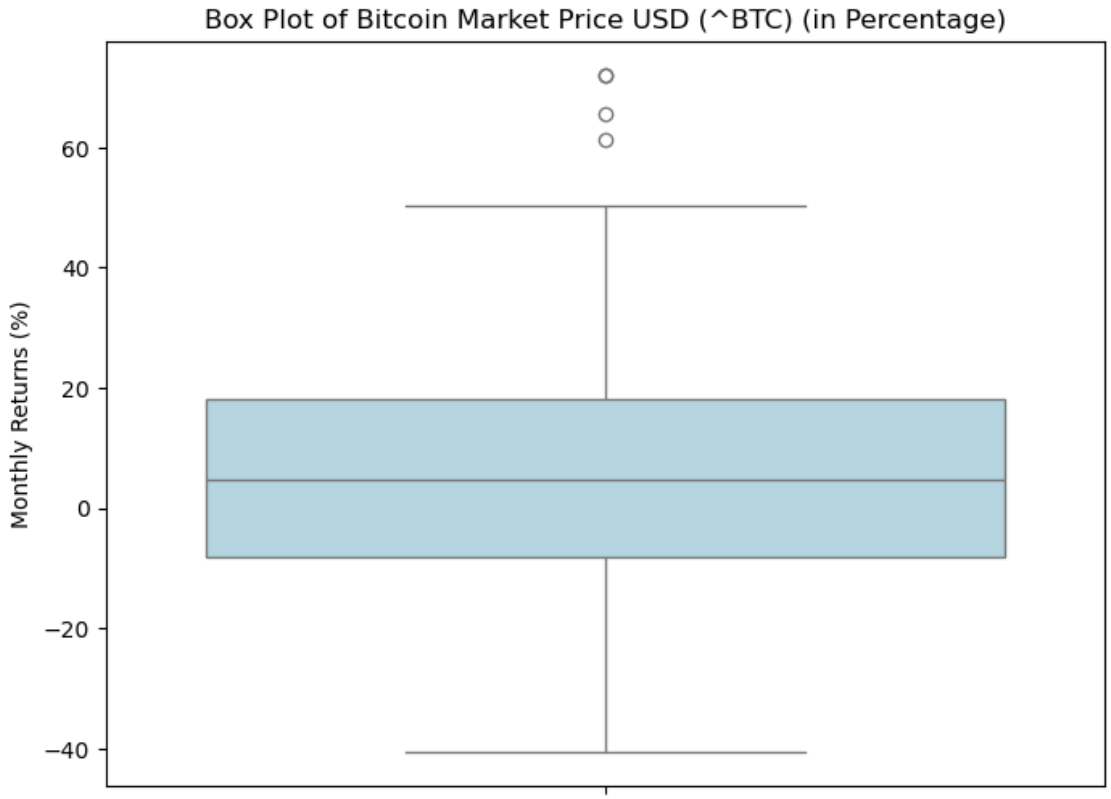
Appendix B1 - Ian Exploratory Data Analysis



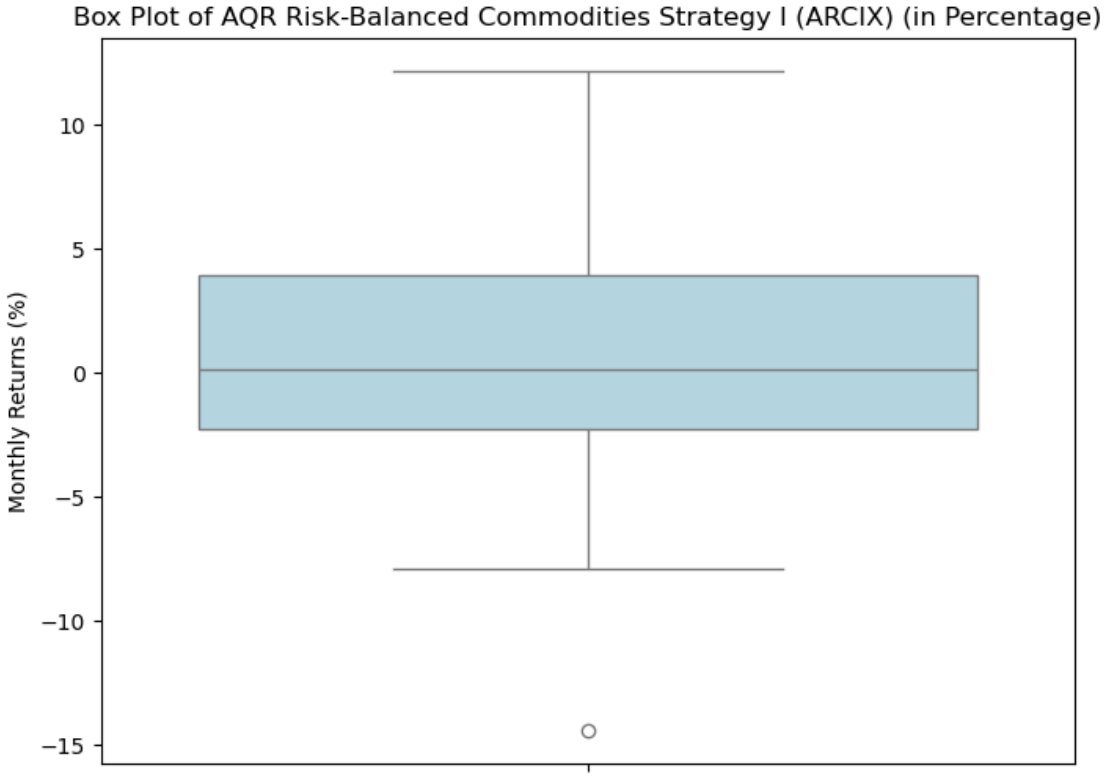


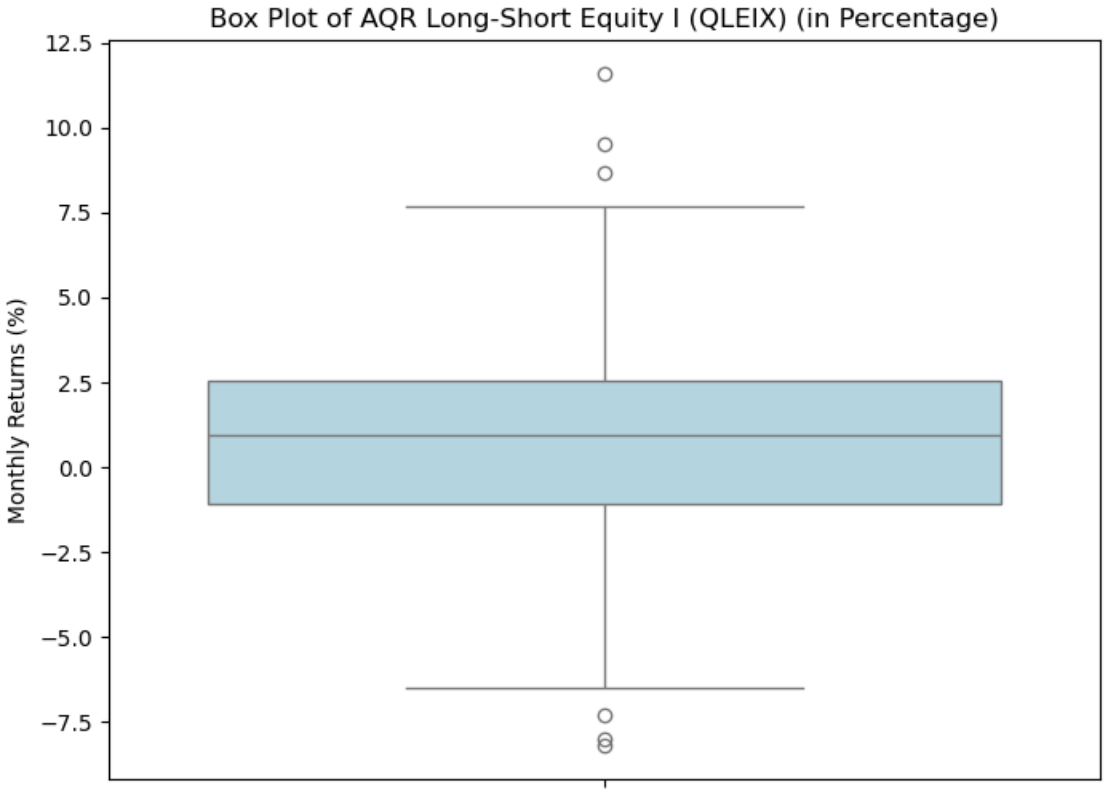
Appendix B1 - Ian Exploratory Data Analysis



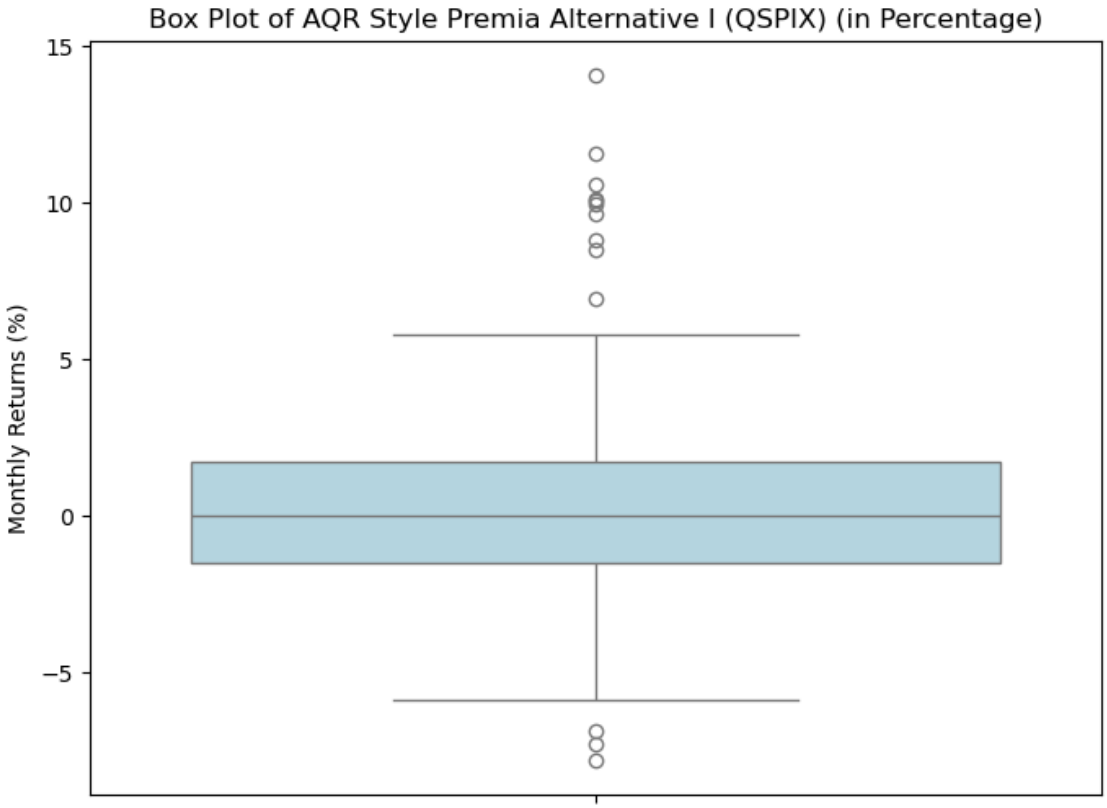


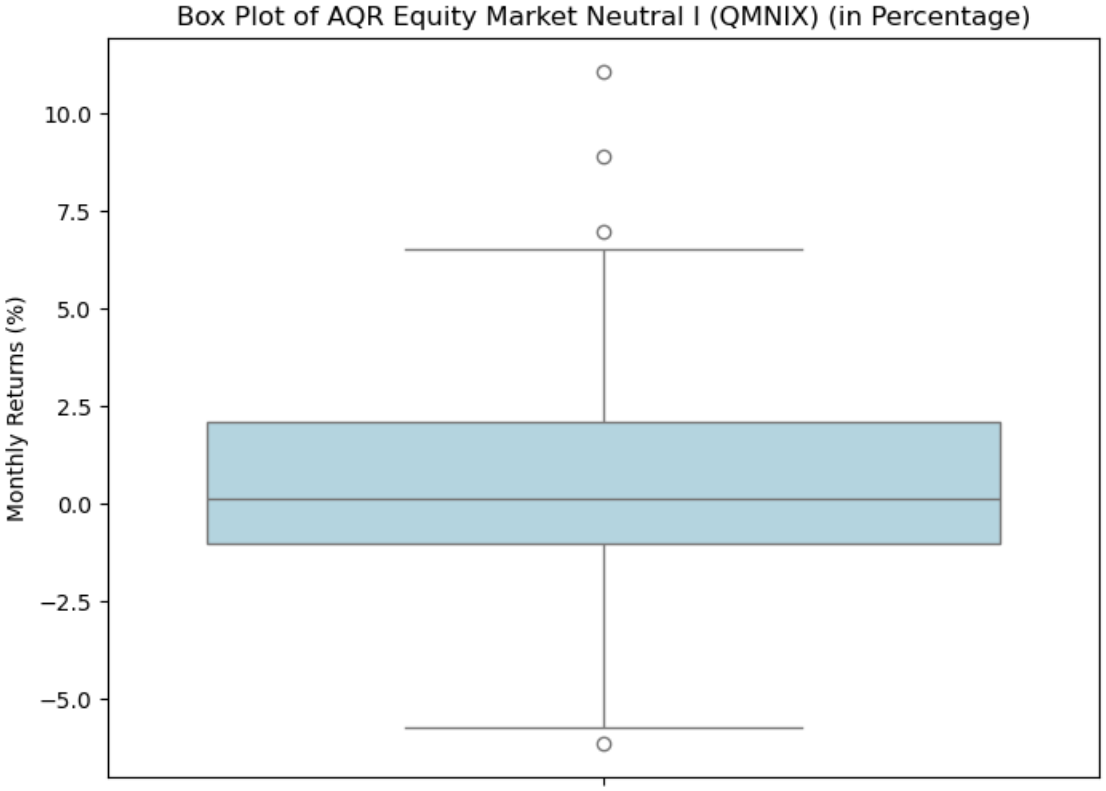
Appendix B1 - Ian Exploratory Data Analysis



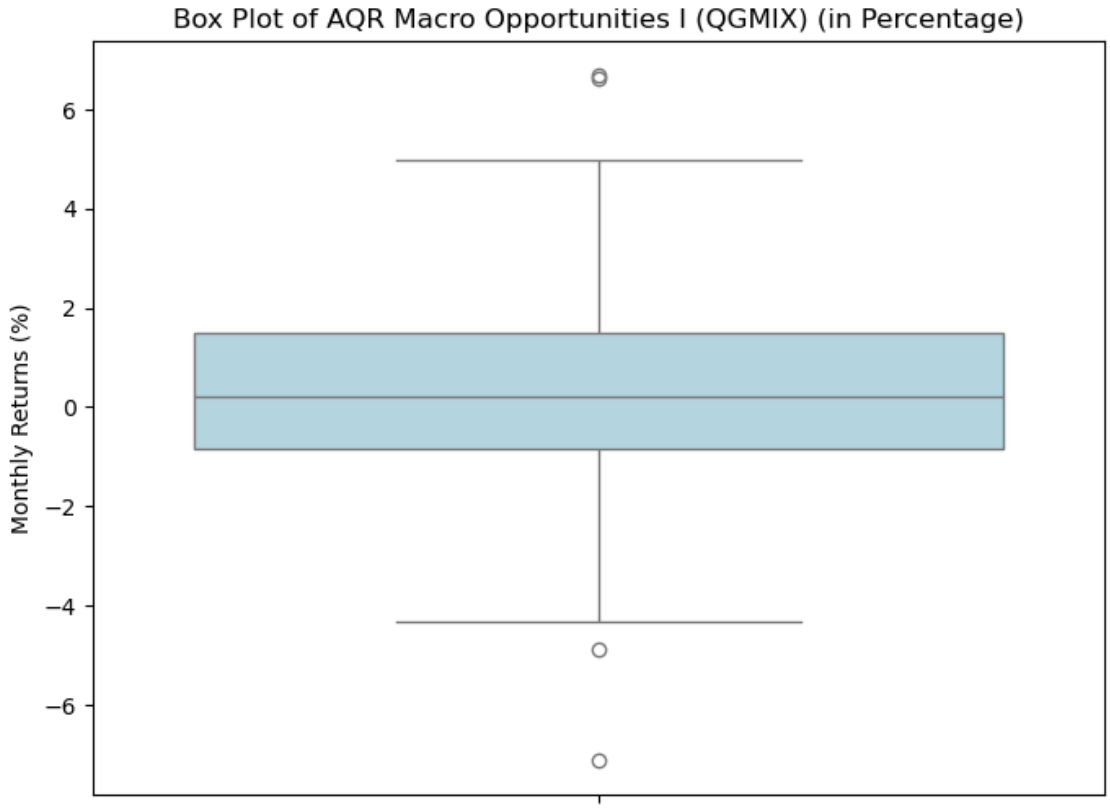


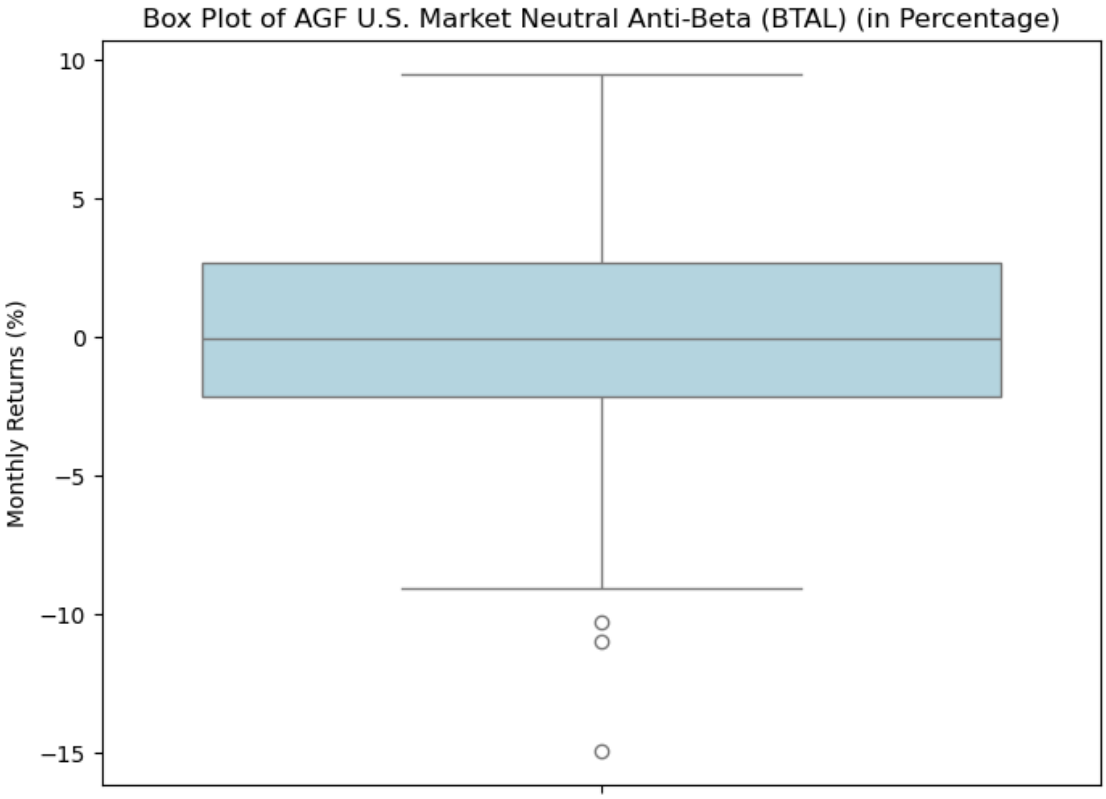
Appendix B1 - Ian Exploratory Data Analysis



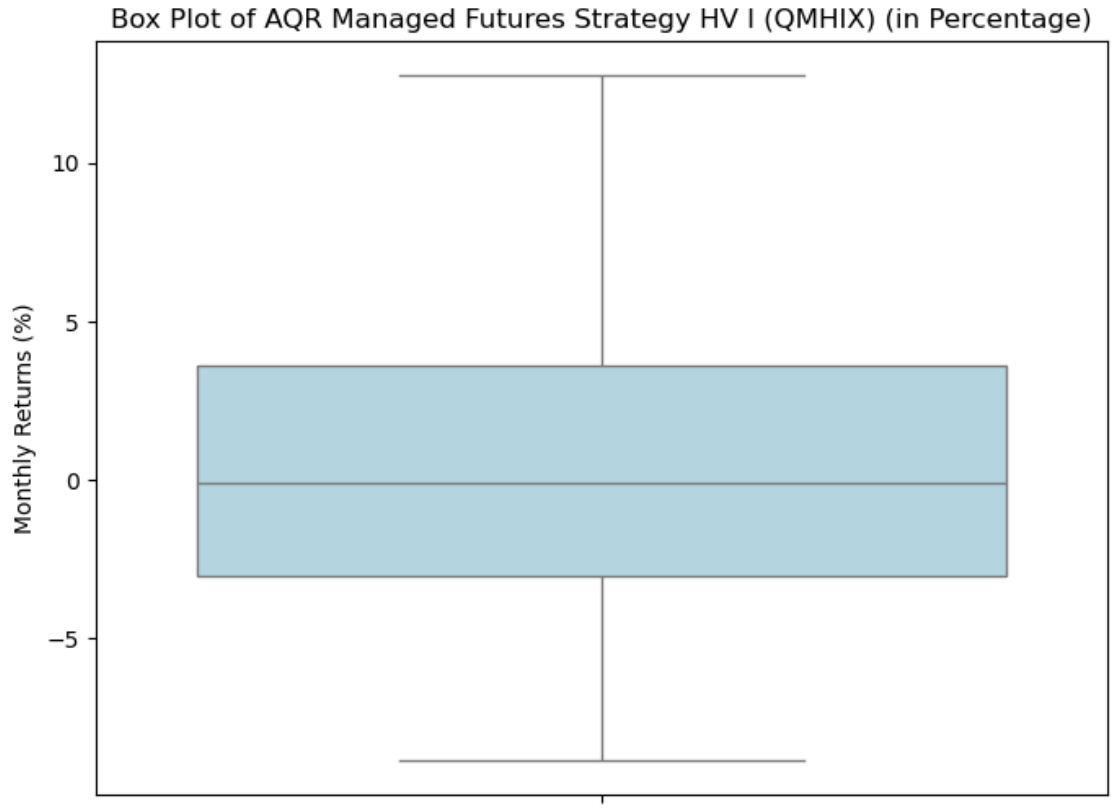


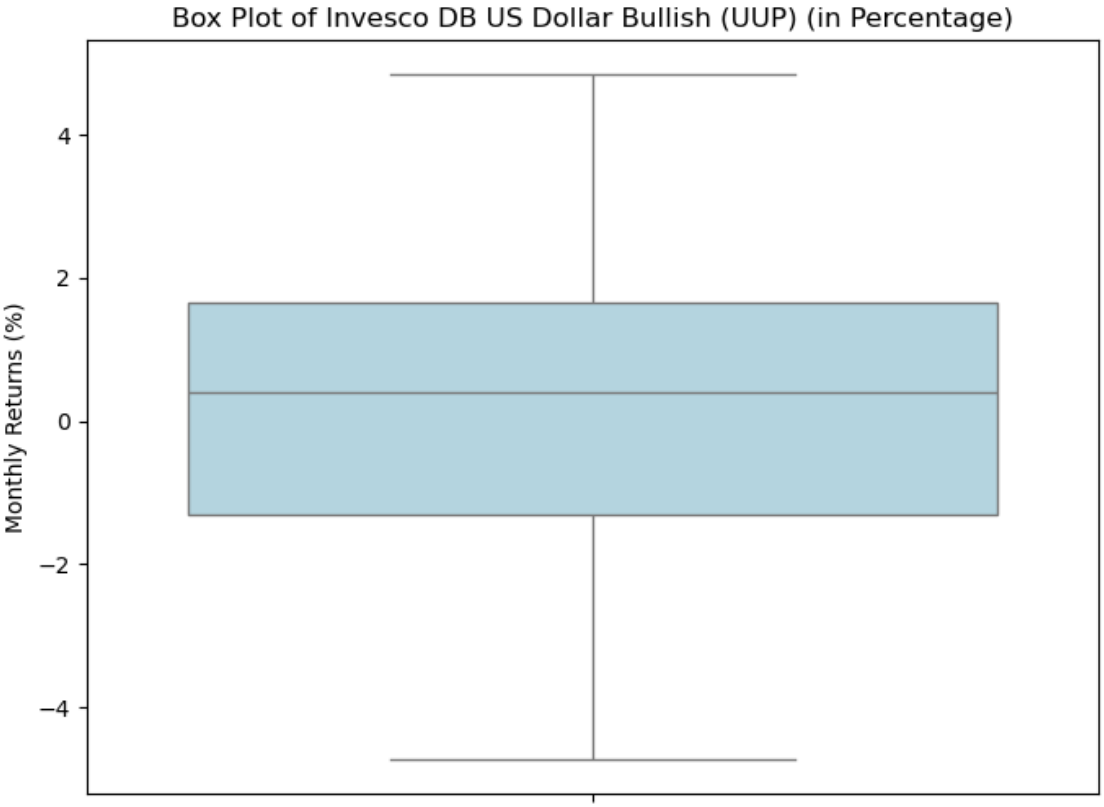
Appendix B1 - Ian Exploratory Data Analysis



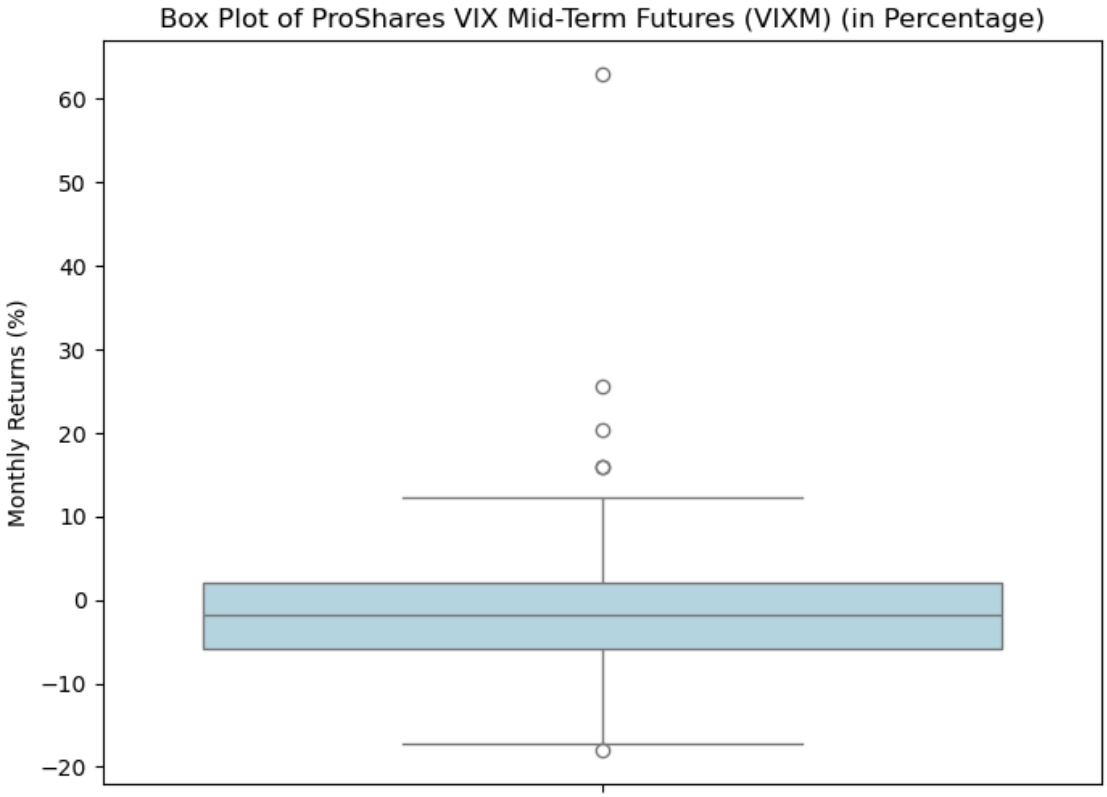


Appendix B1 - Ian Exploratory Data Analysis





Appendix B1 - Ian Exploratory Data Analysis



9 Asset Returns During Benchmark Outlier Months (Table 3)

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.colors as mcolors
import os
import dataframe_image as dfi

# Load the dataset
file_path = 'Opportunity_Set.xlsx'
data = pd.read_excel(file_path)

# Define the output folder name
output_dir = 'OUTLIER_TABLE'
os.makedirs(output_dir, exist_ok=True)

# Extract VASIX returns and dates
vasix_returns = data['Vanguard LifeStrategy Income Fund (VASIX)']
vasix_dates = data['Date']
```

```
# Function to format the date
def format_date(date):
    return date.strftime('%b-%Y')

# Calculate outliers based on IQR logic (1.5 * IQR threshold for detecting␣
 ↪outliers)
Q1 = vasix_returns.quantile(0.25)
Q3 = vasix_returns.quantile(0.75)
IQR = Q3 - Q1
vasix_outliers = vasix_returns[(vasix_returns < (Q1 - 1.5 * IQR)) |␣
 ↪(vasix_returns > (Q3 + 1.5 * IQR))]

# Sort outliers in descending order
vasix_outliers = vasix_outliers.sort_values(ascending=False)

# Get the corresponding dates for the outliers
vasix_outlier_dates = vasix_dates.loc[vasix_outliers.index]

# Split outliers into positive and negative
vasix_outliers_pos = vasix_outliers[vasix_outliers > 0]
vasix_outliers_neg = vasix_outliers[vasix_outliers < 0]

# Get corresponding dates
vasix_outlier_dates_pos = vasix_outlier_dates.loc[vasix_outliers_pos.index]
vasix_outlier_dates_neg = vasix_outlier_dates.loc[vasix_outliers_neg.index]

# Initialize DataFrames for positive and negative outliers
comparative_returns_pos = pd.DataFrame({
    "Date": vasix_outlier_dates_pos.apply(format_date),
    "VASIX Return (%)": vasix_outliers_pos * 100  # Convert to percentage
})

comparative_returns_neg = pd.DataFrame({
    "Date": vasix_outlier_dates_neg.apply(format_date),
    "VASIX Return (%)": vasix_outliers_neg * 100  # Convert to percentage
})

# Dictionary to map formatted column names to original asset names
asset_column_mapping = {}

# Iterate over all assets and capture the return corresponding to VASIX's␣
 ↪outlier periods
for asset in data.columns.drop(['Date', 'Vanguard LifeStrategy Income Fund␣
 ↪(VASIX)']):
    asset_ticker = asset.split('(')[-1].replace(')', '').strip()
```

Appendix B1 - Ian Exploratory Data Analysis

```
    # Positive outliers
    asset_outlier_returns_pos = data.loc[vasix_outliers_pos.index, asset] * 100␣
 ↪ # Convert to percentage
    comparative_returns_pos[f"{asset_ticker}"] = asset_outlier_returns_pos
    asset_column_mapping[asset_ticker] = asset  # Map ticker to original asset␣
 ↪name

    # Negative outliers
    asset_outlier_returns_neg = data.loc[vasix_outliers_neg.index, asset] * 100␣
 ↪ # Convert to percentage
    comparative_returns_neg[f"{asset_ticker}"] = asset_outlier_returns_neg

# Convert all columns to numeric values explicitly to handle potential type␣
 ↪mismatches
comparative_returns_pos.iloc[:, 1:] = comparative_returns_pos.iloc[:, 1:].
 ↪apply(pd.to_numeric, errors='coerce')
comparative_returns_neg.iloc[:, 1:] = comparative_returns_neg.iloc[:, 1:].
 ↪apply(pd.to_numeric, errors='coerce')

# Format the output with two decimal places
comparative_returns_pos.iloc[:, 1:] = comparative_returns_pos.iloc[:, 1:].
 ↪round(2)
comparative_returns_neg.iloc[:, 1:] = comparative_returns_neg.iloc[:, 1:].
 ↪round(2)

# Calculate average returns for positive and negative outliers
average_pos = comparative_returns_pos.iloc[:, 1:].mean()
average_neg = comparative_returns_neg.iloc[:, 1:].mean()

# Create average rows as dictionaries to ensure all columns are populated
average_pos_dict = {"Date": "Average Positive", "VASIX Return (%)":␣
 ↪average_pos["VASIX Return (%)"]}
average_neg_dict = {"Date": "Average Negative", "VASIX Return (%)":␣
 ↪average_neg["VASIX Return (%)"]}

for col in comparative_returns_pos.columns.drop(['Date', 'VASIX Return (%)']):
    average_pos_dict[col] = average_pos[col]
    average_neg_dict[col] = average_neg[col]

# Combine positive and negative outliers
comparative_returns_combined = pd.concat([comparative_returns_pos,␣
 ↪comparative_returns_neg], ignore_index=True)

# Append average rows
comparative_returns_combined = pd.concat([
    comparative_returns_combined,
```

```
    pd.DataFrame([average_pos_dict, average_neg_dict])
], ignore_index=True)

# Compute correlations using all outliers (both positive and negative)
combined_outliers = vasix_outliers
correlations = {}
for asset_ticker in asset_column_mapping:
    correlations[asset_ticker] = combined_outliers.corr(data.
 ↪loc[combined_outliers.index, asset_column_mapping[asset_ticker]])

# Create a row for correlations at the bottom
correlation_data = {"Date": "Correlation", "VASIX Return (%)": 1.0}  #␣
 ↪Correlation of VASIX with itself is 1.0
for asset_ticker, corr_value in correlations.items():
    correlation_data[f"{asset_ticker}"] = corr_value

correlation_row = pd.Series(correlation_data)

# Append the correlation row at the bottom
comparative_returns_combined = pd.concat([comparative_returns_combined, pd.
 ↪DataFrame([correlation_row])], ignore_index=True)

# Rearrange columns to ensure "Date" is first
comparative_returns_combined = comparative_returns_combined[[
    "Date", "VASIX Return (%)"
] + [col for col in comparative_returns_combined.columns if col not in ["Date",␣
 ↪"VASIX Return (%)"]]]

# Identify the indices where separator lines should be added
positive_count = len(comparative_returns_pos)
negative_count = len(comparative_returns_neg)
# Existing separators after positive and negative outliers
separator_indices = [positive_count, positive_count + negative_count]
# Add a second separator above the 3rd row from the bottom
if len(comparative_returns_combined) >= 4:
    additional_separator_index = len(comparative_returns_combined) - 3  #␣
 ↪Correctly targeting the 3rd row from the bottom
    separator_indices.append(additional_separator_index)

# Remove duplicates and sort the separator indices
separator_indices = sorted(list(set(separator_indices)))

# Apply color highlights and add separator lines using a valid color map
def highlight_table(row):
    styles = [''] * len(row)  # Initialize with empty styles

    # Apply separator lines
```

```
    if row.name in separator_indices:
        # Apply a top border to this row
        styles = [s + 'border-top: 2px solid black;' for s in styles]

    # Highlight correlation row
    if row["Date"] == "Correlation":
        # Normalize values between -1 and 1 for highlighting
        norm = mcolors.Normalize(vmin=-1, vmax=1)
        cmap = sns.diverging_palette(240, 10, as_cmap=True)  # Blue to red
        # Apply colors to the correlation row, leaving the Date column blank
        for i, col in enumerate(row.index):
            if col == "Date":
                continue
            try:
                val = float(row[col])
                color = mcolors.to_hex(cmap(norm(val)))
                styles[i] += f"background-color: {color};"
            except:
                styles[i] += ''

    # Highlight average rows
    if row["Date"] in ["Average Positive", "Average Negative"]:
        # Apply a light gray background
        styles = [s + 'background-color: #f0f0f0;' for s in styles]

    return styles

# Bold the "Average" and "Correlation" rows for emphasis
def bold_rows(row):
    styles = [''] * len(row)
    if row["Date"] in ["Average Positive", "Average Negative", "Correlation"]:
        styles = [s + 'font-weight: bold;' for s in styles]
    return styles

# Apply the color highlighting and bolding, then format the table
styled_table = comparative_returns_combined.style.apply(highlight_table,␣
 ↪axis=1)\
                                                    .apply(bold_rows, axis=1)\
                                                    .format({
                                                        "VASIX Return (%)": "{:.2f}",
                                                        **{col: "{:.2f}" for col in␣
 ↪comparative_returns_combined.columns if col not in ["Date", "VASIX Return␣
 ↪(%)"]}
                                                    })

# Save the styled table as a PNG using dataframe_image
output_path = os.path.join(output_dir, 'styled_table.png')
```

```
dfi.export(styled_table, output_path)

# Display the styled table in Jupyter Notebook
styled_table
```

[13]: <pandas.io.formats.style.Styler at 0x13eb89700>

10 Asset Risk-Return Characteristics (Table 4)

[19]:
```
import numpy as np
import pandas as pd
from scipy.stats import skew, kurtosis, t
import os

# Load the data
file_path = 'Opportunity_Set.xlsx'
data = pd.read_excel(file_path)

# Ensure the returns data has a DateTimeIndex
data['Date'] = pd.to_datetime(data['Date'])  # Convert 'Date' column to datetime
data.set_index('Date', inplace=True)  # Set 'Date' as the index

# Helper functions for calculation
def calculate_cagr(returns, periods_per_year):
    total_return = (1 + returns).prod()
    n_years = len(returns) / periods_per_year
    cagr = total_return ** (1 / n_years) - 1
    return cagr

def calculate_annualized_std(returns, periods_per_year):
    return returns.std() * np.sqrt(periods_per_year)

def calculate_annual_returns(returns, periods_per_year):
    annual_returns = (1 + returns).resample('YE').prod() - 1  # Use 'YE' for␣
 ↪year end
    best_year = annual_returns.max()
    worst_year = annual_returns.min()
    return best_year, worst_year

def calculate_max_drawdown(returns):
    cumulative_returns = (1 + returns).cumprod()
    peak = cumulative_returns.cummax()
    drawdown = (cumulative_returns - peak) / peak
    max_drawdown = drawdown.min()
    return max_drawdown
```

```
def calculate_sharpe_ratio(returns, risk_free_rate, periods_per_year):
    excess_returns = returns - (risk_free_rate / periods_per_year)
    annualized_sharpe = (excess_returns.mean() / excess_returns.std()) * np.
 ↪sqrt(periods_per_year)
    return annualized_sharpe

def calculate_sortino_ratio(returns, risk_free_rate, periods_per_year):
    excess_returns = returns - (risk_free_rate / periods_per_year)
    downside_returns = returns[returns < 0]
    downside_std = downside_returns.std() * np.sqrt(periods_per_year)
    sortino_ratio = (excess_returns.mean() / downside_std) * np.
 ↪sqrt(periods_per_year)
    return sortino_ratio

# Assuming 12 periods per year (monthly data)
periods_per_year = 12
risk_free_rate = 0.02

# Create an empty list to store the results for each asset
results_list = []

for asset in data.columns:
    asset_returns = data[asset].dropna()  # Drop NaN values for each asset

    # Calculate metrics for the asset
    cagr = calculate_cagr(asset_returns, periods_per_year)
    annualized_std = calculate_annualized_std(asset_returns, periods_per_year)
    best_year, worst_year = calculate_annual_returns(asset_returns,␣
 ↪periods_per_year)
    max_drawdown = calculate_max_drawdown(asset_returns)
    sharpe_ratio = calculate_sharpe_ratio(asset_returns, risk_free_rate,␣
 ↪periods_per_year)
    sortino_ratio = calculate_sortino_ratio(asset_returns, risk_free_rate,␣
 ↪periods_per_year)

    # Append the results to the list
    results_list.append({
        "Asset": asset,
        "CAGR": f"{100*cagr:.2f}",
        "Annualized Std Dev": f"{100*annualized_std:.2f}",
        "Best Year": f"{100*best_year:.2f}",
        "Worst Year": f"{100*worst_year:.2f}",
        "Max Drawdown": f"{100*max_drawdown:.2f}",
        "Sharpe Ratio": f"{sharpe_ratio:.2f}",
        "Sortino Ratio": f"{sortino_ratio:.2f}"
    })
```

```
# Convert the results to a DataFrame
results_df = pd.DataFrame(results_list)

# Modify the column headers with line breaks and ensure equal width columns␣
 ↪(except Asset)
column_headers = {
    "CAGR": "CAGR\n(%)",
    "Annualized Std Dev": "Annualized\nStd Dev\n(%)",
    "Best Year": "Best Year\n(%)",
    "Worst Year": "Worst Year\n(%)",
    "Max Drawdown": "Max\nDrawdown\n(%)",
    "Sharpe Ratio": "Sharpe\nRatio",
    "Sortino Ratio": "Sortino\nRatio"
}
results_df.rename(columns=column_headers, inplace=True)

# Center-align and align decimals in numeric columns, no text wrapping in Asset␣
 ↪column
styled_table = results_df.style.set_properties(
    subset=['CAGR\n(%)', 'Annualized\nStd Dev\n(%)', 'Best Year\n(%)', 'Worst␣
 ↪Year\n(%)',
            'Max\nDrawdown\n(%)', 'Sharpe\nRatio', 'Sortino\nRatio'],
    **{'text-align': 'right', 'padding-right': '60px'}  # More padding for␣
 ↪better centering
).set_table_styles([
    {'selector': 'th', 'props': [('text-align', 'center'), ('width', '100px')]},
    {'selector': 'th.col0', 'props': [('text-align', 'right'), ('white-space',␣
 ↪'nowrap'), ('width', '250px')]},  # Asset column no wrapping
    {'selector': 'td, th', 'props': [('border', '1px solid black')]},
]).format(precision=2).set_caption("Risk and Return Characteristics for All␣
 ↪Assets")

# Apply formatting for negative values in red font using map instead of applymap
styled_table = styled_table.map(
    lambda x: 'color: red' if '-' in str(x) else 'color: black',
    subset=['Worst Year\n(%)', 'Max\nDrawdown\n(%)']
)

# Output folder setup
output_folder = 'RISK-RETURN_CHARACTERISTICS_TABLE'
os.makedirs(output_folder, exist_ok=True)
output_file_path = os.path.join(output_folder,␣
 ↪"risk_return_characteristics_table.png")

# Save the styled table as an image
try:
```

```
    import df2img
    fig = df2img.plot_dataframe(styled_table, title="Risk and Return␣
 ↪Characteristics", fontsize=14)
    df2img.save_dataframe(fig=fig, filename=output_file_path)
except ImportError:
    print("df2img is required to export the styled table as an image")
except Exception as e:
    print(f"An error occurred while saving the image: {e}")

# Display the styled table in Jupyter Notebook
styled_table
```

```
df2img is required to export the styled table as an image
```

[19]: <pandas.io.formats.style.Styler at 0x31c4051c0>

11 Asset Regressions Against the Benchmark (VASIX) (Table 5)

[21]:
```
import numpy as np
import pandas as pd
from scipy import stats
import os
import statsmodels.api as sm

# Load the data
file_path = 'Opportunity_Set.xlsx'
data = pd.read_excel(file_path)

# Ensure the returns data has a DateTimeIndex
data['Date'] = pd.to_datetime(data['Date'])  # Convert 'Date' column to datetime
data.set_index('Date', inplace=True)  # Set 'Date' as the index

# Regression analysis for each asset against VASIX
benchmark = data['Vanguard LifeStrategy Income Fund (VASIX)'].dropna()
regression_results = []

for asset in data.columns:
    if asset != 'Vanguard LifeStrategy Income Fund (VASIX)':
        asset_returns = data[asset].dropna()
        combined_data = pd.concat([benchmark, asset_returns], axis=1).dropna()
        X = combined_data['Vanguard LifeStrategy Income Fund (VASIX)']
        Y = combined_data[asset]
        X = sm.add_constant(X)  # Add intercept term
        model = sm.OLS(Y, X).fit()

        intercept = model.params['const'] * 12 * 100  # Annualized intercept
        intercept_tstat = model.tvalues['const']
```

```
        intercept_pvalue = model.pvalues['const']
        beta = model.params['Vanguard LifeStrategy Income Fund (VASIX)']
        beta_tstat = model.tvalues['Vanguard LifeStrategy Income Fund (VASIX)']
        beta_pvalue = model.pvalues['Vanguard LifeStrategy Income Fund (VASIX)']

        regression_results.append({
            "Asset": asset,
            "R2": model.rsquared,
            "Intercept (Annualized)": intercept,
            "Intercept t-stat": intercept_tstat,
            "Intercept p-value": intercept_pvalue,
            "Beta": beta,
            "Beta t-stat": beta_tstat,
            "Beta p-value": beta_pvalue
        })

# Convert the regression results to a DataFrame
regression_df = pd.DataFrame(regression_results)

# Function to style significant values
def highlight_significant(row):
    styles = []
    # Highlight Intercept columns if Intercept p-value < 0.05
    if row['Intercept p-value'] < 0.05:
        styles.extend(['font-weight: bold'] * 3)  # For 'Intercept␣
 ↪(Annualized)', 'Intercept t-stat', 'Intercept p-value'
    else:
        styles.extend([''] * 3)

    # Highlight Beta columns if Beta p-value < 0.05
    if row['Beta p-value'] < 0.05:
        styles.extend(['font-weight: bold'] * 3)  # For 'Beta', 'Beta t-stat',␣
 ↪'Beta p-value'
    else:
        styles.extend([''] * 3)

    return styles  # Total of 6 styles

# Styling functions for different column groups
def style_intercept(val):
    return 'background-color: #e6f2ff; border-right: 2px solid #000'  # Light␣
 ↪blue with right border

def style_beta(val):
    return 'background-color: #e6ffe6; border-right: 2px solid #000'  # Light␣
 ↪green with right border
```

```
def style_r2(val):
    return 'background-color: #ffffe6'  # Light yellow

# Style the regression table
styled_regression_table = (
    regression_df.style
    .format(
        {
            "Intercept (Annualized)": "{:.2f}",
            "Intercept t-stat": "{:.2f}",
            "Intercept p-value": "{:.4f}",
            "Beta": "{:.2f}",
            "Beta t-stat": "{:.2f}",
            "Beta p-value": "{:.4f}",
            "R2": "{:.2f}"
        }
    )
    .apply(
        highlight_significant,
        axis=1,
        subset=['Intercept (Annualized)', 'Intercept t-stat', 'Intercept␣
↪p-value',
                'Beta', 'Beta t-stat', 'Beta p-value']
    )
    .map(
        style_intercept,
        subset=['Intercept (Annualized)', 'Intercept t-stat', 'Intercept␣
↪p-value']
    )
    .map(
        style_beta,
        subset=['Beta', 'Beta t-stat', 'Beta p-value']
    )
    .map(
        style_r2,
        subset=['R2']
    )
    .set_properties(
        subset=['R2', 'Intercept (Annualized)', 'Intercept t-stat', 'Intercept␣
↪p-value',
                'Beta', 'Beta t-stat', 'Beta p-value'],
        **{'text-align': 'center'}
    )
    .set_table_styles([
        {'selector': 'th', 'props': [('text-align', 'center')]},
        {'selector': 'td, th', 'props': [('border', '1px solid black')]},
```

```
        {'selector': 'td.col0', 'props': [('text-align', 'left'),␣
 ↪('white-space', 'nowrap')]},  # Asset column no wrapping
    ])
    .set_caption("Regression Results Against Benchmark (VASIX)")
)

# Output folder setup for regression results
regression_output_folder = 'ASSET_REGRESSION_RESULTS'
os.makedirs(regression_output_folder, exist_ok=True)
regression_output_file_path = os.path.join(regression_output_folder,␣
 ↪"asset_regression_table.png")

# Save the styled regression table as an image
try:
    import dataframe_image as dfi
    dfi.export(styled_regression_table, regression_output_file_path)
except ImportError:
    print("dataframe_image is required to export the styled regression table as␣
 ↪an image")

# Display the styled regression table in Jupyter Notebook
styled_regression_table
```

[21]: <pandas.io.formats.style.Styler at 0x31d4c57f0>

12 Non-Parametric Bootstrap Results (10,000 Iterations) (Table 6)

[27]:
```
# NON-PARAMETRIC BOOTSTRAP OF ANNUALIZED MEANS AND STANDARD DEVIATIONS (WITH␣
 ↪ESTIMATED AND ACTUAL CAGR)

import os
import numpy as np
import pandas as pd
import dataframe_image as dfi  # For exporting DataFrames as images

# Comment-based snippet name for the folder
snippet_name = "BOOTSTRAP_RESULTS_TABLE"

# Create the directory to save the table
output_folder = snippet_name
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# Exclude the 'Date' column from the data
asset_columns = data.columns.drop('Date', errors='ignore')
```

```
# Determine the number of periods per year based on data frequency
periods_per_year = 12  # Adjust this value according to your data frequency

# Initialize dictionary to store results
bootstrap_results_dict = {}

# Set the number of bootstrap iterations
n_iterations = 10000  # You can adjust this as needed
np.random.seed(42)  # Set seed for reproducibility

# Loop through each asset in the dataset (excluding the 'Date' column)
for asset in asset_columns:
    # Get the asset returns
    asset_returns = data[asset].dropna()
    n_size = len(asset_returns)

    # Arrays to hold bootstrap estimates
    bootstrap_means = np.zeros(n_iterations)
    bootstrap_vars = np.zeros(n_iterations)

    # Perform bootstrap resampling
    for i in range(n_iterations):
        # Generate a bootstrap sample with replacement
        bootstrap_sample = np.random.choice(asset_returns, size=n_size,␣
↪replace=True)

        # Calculate mean and variance for the bootstrap sample
        bootstrap_means[i] = np.mean(bootstrap_sample)
        bootstrap_vars[i] = np.var(bootstrap_sample, ddof=1)

    # Annualize the bootstrap means and variances
    annualized_bootstrap_means = bootstrap_means * periods_per_year
    annualized_bootstrap_vars = bootstrap_vars * periods_per_year

    # Convert variances to standard deviations
    annualized_bootstrap_std_devs = np.sqrt(annualized_bootstrap_vars)

    # Calculate 95% confidence intervals for the annualized mean returns
    mean_ci_lower, mean_ci_upper = np.percentile(annualized_bootstrap_means, [2.
↪5, 97.5])

    # Calculate 95% confidence intervals for the annualized standard deviations
    std_ci_lower, std_ci_upper = np.percentile(annualized_bootstrap_std_devs,␣
↪[2.5, 97.5])

    # Estimated CAGR calculation using annualized mean and standard deviation
```

```
    estimated_cagr = np.mean(annualized_bootstrap_means) - 0.5 * (np.
↪mean(annualized_bootstrap_std_devs) ** 2)

    # Actual CAGR calculation
    cumulative_return = (1 + asset_returns).prod() - 1
    n_years = len(asset_returns) / periods_per_year
    actual_cagr = (1 + cumulative_return) ** (1 / n_years) - 1

    # Store the results in a dictionary and multiply by 100 to convert to␣
↪percentages
    bootstrap_results_dict[asset] = {
        "Annualized Mean Estimate": np.mean(annualized_bootstrap_means) * 100,
        "Mean 95% CI Lower": mean_ci_lower * 100,
        "Mean 95% CI Upper": mean_ci_upper * 100,
        "Annualized Std Dev Estimate": np.mean(annualized_bootstrap_std_devs) *␣
↪100,
        "Std Dev 95% CI Lower": std_ci_lower * 100,
        "Std Dev 95% CI Upper": std_ci_upper * 100,
        "Estimated CAGR (%)": estimated_cagr * 100,
        "Actual CAGR (%)": actual_cagr * 100
    }

# Convert the results to a DataFrame for better readability
bootstrap_results_df = pd.DataFrame(bootstrap_results_dict).T
bootstrap_results_df.columns = ["Annualized Mean Estimate", "Mean 95% CI␣
↪Lower", "Mean 95% CI Upper",
                                "Annualized Std Dev Estimate", "Std Dev 95% CI␣
↪Lower", "Std Dev 95% CI Upper",
                                "Estimated CAGR (%)", "Actual CAGR (%)"]

# Display the results in a nicely formatted table with percentage formatting␣
↪(two decimal places)
bootstrap_results_styled = bootstrap_results_df.style.format({
    "Annualized Mean Estimate": "{:.2f}",
    "Mean 95% CI Lower": "{:.2f}",
    "Mean 95% CI Upper": "{:.2f}",
    "Annualized Std Dev Estimate": "{:.2f}",
    "Std Dev 95% CI Lower": "{:.2f}",
    "Std Dev 95% CI Upper": "{:.2f}",
    "Estimated CAGR (%)": "{:.2f}",
    "Actual CAGR (%)": "{:.2f}"
}).set_caption("Bootstrap Results for Annualized Asset Returns (Means, Std␣
↪Devs, Estimated, and Actual CAGRs)")

# Center align all data cells
bootstrap_results_styled = bootstrap_results_styled.set_properties(
```

```
    **{'text-align': 'center'},
    subset=pd.IndexSlice[:, :]
)

# Apply styles to the table
bootstrap_results_styled = bootstrap_results_styled.set_table_styles(
    [
        # Style the header
        {'selector': 'th', 'props': [('text-align', 'center'), ('font-weight',␣
 ↪'bold')]},
        # Style the index (asset names) to be right-aligned and prevent wrapping
        {'selector': 'th.row_heading', 'props': [
            ('text-align', 'right'),
            ('white-space', 'nowrap'),
            ('font-weight', 'bold')
        ]}
    ]
)

# Save the formatted table as a PNG file inside the created folder
dfi.export(bootstrap_results_styled, os.path.join(output_folder,␣
 ↪"bootstrap_results_table.png"))

# Optionally, display the formatted table in Jupyter
bootstrap_results_styled
```

[27]: <pandas.io.formats.style.Styler at 0x309266810>

13 Non-Parametric Bootstrap Results Sampled from Lowest & Highest Quartiles (10,000 Iterations) (Not an Exhibit in the Report)

[28]:
```
# NON-PARAMETRIC BOOTSTRAP OF ANNUALIZED MEANS AND STANDARD DEVIATIONS (USING␣
 ↪LOWEST AND HIGHEST QUARTILES)

import os
import numpy as np
import pandas as pd
import dataframe_image as dfi  # For exporting DataFrames as images

# Comment-based snippet name for the folder
snippet_name = "BOOTSTRAP_TAILS_RESULTS_TABLE"  # Updated folder name

# Create the directory to save the table
output_folder = snippet_name
```

```
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# Ensure the 'Date' column is excluded if present
asset_columns = data.columns.drop('Date', errors='ignore')

# Determine the number of periods per year based on data frequency
periods_per_year = 12  # Adjust this value according to your data frequency

# Initialize dictionary to store results
bootstrap_results_dict = {}

# Set the number of bootstrap iterations
n_iterations = 10000  # You can adjust this as needed
np.random.seed(42)  # Set seed for reproducibility

# Loop through each asset in the dataset (excluding the 'Date' column if␣
 ↪necessary)
for asset in asset_columns:
    # Get the asset returns
    asset_returns = data[asset].dropna()
    n_size = len(asset_returns)

    # Calculate the lower (Q1) and upper (Q3) quartiles
    Q1 = asset_returns.quantile(0.25)
    Q3 = asset_returns.quantile(0.75)

    # Filter the data to only include values in the lowest (Q1) and highest␣
 ↪quartiles (values greater than Q3)
    tail_returns = asset_returns[(asset_returns <= Q1) | (asset_returns >= Q3)]

    # Ensure tail_returns contains numeric data
    tail_returns = pd.to_numeric(tail_returns, errors='coerce').dropna()

    # Arrays to hold bootstrap estimates
    bootstrap_means = np.zeros(n_iterations)
    bootstrap_vars = np.zeros(n_iterations)

    # Perform bootstrap resampling
    for i in range(n_iterations):
        # Generate a bootstrap sample with replacement from the tail data
        bootstrap_sample = np.random.choice(tail_returns, size=n_size,␣
 ↪replace=True)

        # Calculate mean and variance for the bootstrap sample
        bootstrap_means[i] = np.mean(bootstrap_sample)
        bootstrap_vars[i] = np.var(bootstrap_sample, ddof=1)
```

```
    # Annualize the bootstrap means and variances
    annualized_bootstrap_means = bootstrap_means * periods_per_year
    annualized_bootstrap_vars = bootstrap_vars * periods_per_year

    # Convert variances to standard deviations
    annualized_bootstrap_std_devs = np.sqrt(annualized_bootstrap_vars)

    # Calculate 95% confidence intervals for the annualized mean returns
    mean_ci_lower, mean_ci_upper = np.percentile(annualized_bootstrap_means, [2.
↪5, 97.5])

    # Calculate 95% confidence intervals for the annualized standard deviations
    std_ci_lower, std_ci_upper = np.percentile(annualized_bootstrap_std_devs,␣
↪[2.5, 97.5])

    # Estimate the CAGR using the annualized mean and standard deviation
    estimated_cagr = (np.mean(annualized_bootstrap_means) - 0.5 * (np.
↪mean(annualized_bootstrap_std_devs) ** 2)) * 100

    # Actual CAGR calculation
    cumulative_return = (1 + asset_returns).prod() - 1
    n_years = len(asset_returns) / periods_per_year
    actual_cagr = (1 + cumulative_return) ** (1 / n_years) - 1

    # Store the results in a dictionary and multiply by 100 to convert to␣
↪percentages
    bootstrap_results_dict[asset] = {
        "Annualized Mean Estimate": np.mean(annualized_bootstrap_means) * 100,
        "Mean 95% CI Lower": mean_ci_lower * 100,
        "Mean 95% CI Upper": mean_ci_upper * 100,
        "Annualized Std Dev Estimate": np.mean(annualized_bootstrap_std_devs) *␣
↪100,
        "Std Dev 95% CI Lower": std_ci_lower * 100,
        "Std Dev 95% CI Upper": std_ci_upper * 100,
        "Estimated CAGR (%)": estimated_cagr,
        "Actual CAGR (%)": actual_cagr * 100
    }

# Convert the results to a DataFrame for better readability
bootstrap_results_df = pd.DataFrame(bootstrap_results_dict).T
bootstrap_results_df.columns = ["Annualized Mean Estimate", "Mean 95% CI␣
↪Lower", "Mean 95% CI Upper",
                                "Annualized Std Dev Estimate", "Std Dev 95% CI␣
↪Lower", "Std Dev 95% CI Upper",
                                "Estimated CAGR (%)", "Actual CAGR (%)"]
```

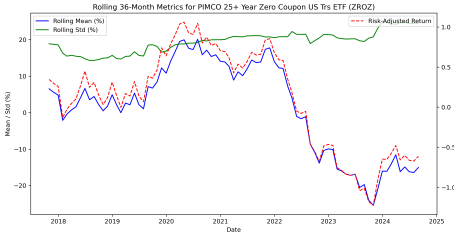
```
# Display the results in a nicely formatted table with percentage formatting␣
 ↪(two decimal places)
bootstrap_results_styled = bootstrap_results_df.style.format({
    "Annualized Mean Estimate": "{:.2f}",
    "Mean 95% CI Lower": "{:.2f}",
    "Mean 95% CI Upper": "{:.2f}",
    "Annualized Std Dev Estimate": "{:.2f}",
    "Std Dev 95% CI Lower": "{:.2f}",
    "Std Dev 95% CI Upper": "{:.2f}",
    "Estimated CAGR (%)": "{:.2f}",
    "Actual CAGR (%)": "{:.2f}"
}).set_caption("Bootstrap Results for Annualized Asset Returns (Means, Std␣
 ↪Devs, and CAGRs)")

# Center align all data cells
bootstrap_results_styled = bootstrap_results_styled.set_properties(
    **{'text-align': 'center'},
    subset=pd.IndexSlice[:, :]
)

# Apply styles to the table
bootstrap_results_styled = bootstrap_results_styled.set_table_styles(
    [
        # Style the header
        {'selector': 'th', 'props': [('text-align', 'center'), ('font-weight',␣
 ↪'bold')]},
        # Style the index (asset names) to be right-aligned and prevent wrapping
        {'selector': 'th.row_heading', 'props': [
            ('text-align', 'right'),
            ('white-space', 'nowrap'),
            ('font-weight', 'bold')
        ]}
    ]
)

# Save the formatted table as a PNG file inside the created folder
dfi.export(bootstrap_results_styled, os.path.join(output_folder,␣
 ↪"bootstrap_tails_results_table.png"))

# Optionally, display the formatted table in Jupyter
bootstrap_results_styled
```

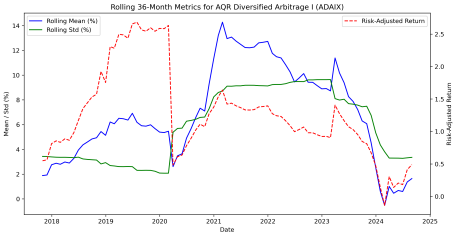
[28]: <pandas.io.formats.style.Styler at 0x169df04d0>

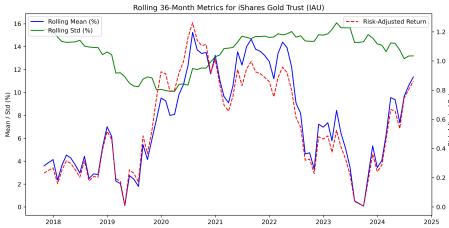
14 Rolling 36-Month Means, Standard Deviations, and Risk-Adj Returns (Figure 7)

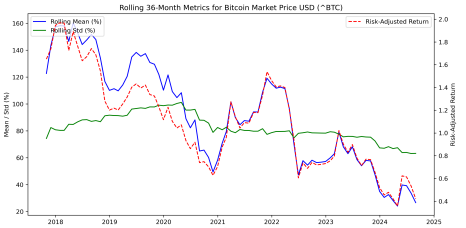
[22]:
```
import os
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from PIL import Image
from IPython.display import display

# Load the dataset
file_path = 'Opportunity_Set.xlsx'
data = pd.read_excel(file_path)

# Check if 'Date' is already the index or if the column exists with a different␣
 ↪name
if 'Date' in data.columns:
    data = data.set_index('Date')  # Set 'Date' as the index if it isn't already

# Create a folder based on the first comment
output_folder = "ROLLING_36_MONTH_MEAN_STD_RISKADJ_V2"
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# Function to plot rolling metrics for a single asset
def plot_rolling_metrics(asset_returns, asset_name, window_size=36):
    # Calculate rolling statistics
    rolling_mean_annualized = asset_returns.rolling(window=window_size).mean()␣
 ↪* 12 * 100  # Annualize and convert to percentage
    rolling_std_annualized = asset_returns.rolling(window=window_size).std() *␣
 ↪(12 ** 0.5) * 100  # Annualize and convert to percentage
    rolling_riskadj_annualized = rolling_mean_annualized /␣
 ↪rolling_std_annualized

    # Drop NaN values dynamically (where rolling data starts)
    rolling_mean_annualized = rolling_mean_annualized.dropna()
    rolling_std_annualized = rolling_std_annualized.dropna()
    rolling_riskadj_annualized = rolling_riskadj_annualized.dropna()

    # Create and save combined rolling metrics plot
    fig, ax1 = plt.subplots(figsize=(12, 6))

    ax1.plot(rolling_mean_annualized.index, rolling_mean_annualized,␣
 ↪color='blue', label='Rolling Mean (%)')
    ax1.plot(rolling_std_annualized.index, rolling_std_annualized,␣
 ↪color='green', label='Rolling Std (%)')
```

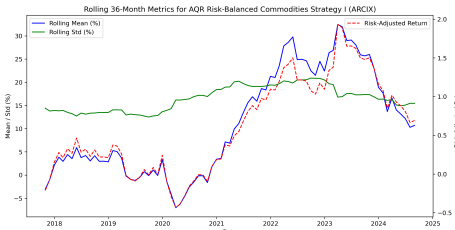
```
    ax1.set_ylabel("Mean / Std (%)")
    ax1.set_xlabel("Date")
    ax1.tick_params(axis='y')
    ax1.legend(loc='upper left')

    ax2 = ax1.twinx()  # Create a second y-axis for risk-adjusted return
    ax2.plot(rolling_riskadj_annualized.index, rolling_riskadj_annualized,␣
 ↪color='red', linestyle='--', label='Risk-Adjusted Return')
    ax2.set_ylabel("Risk-Adjusted Return")
    ax2.tick_params(axis='y')
    ax2.legend(loc='upper right')

    plt.title(f"Rolling 36-Month Metrics for {asset_name}")
    plt.savefig(os.path.join(output_folder, f"{asset_name}_rolling_metrics.
 ↪png"), bbox_inches='tight', dpi=300)
    plt.close()

# Loop through each asset and generate plots
for asset in data.columns:
    asset_returns = data[asset].dropna()  # Drop NaN values
    plot_rolling_metrics(asset_returns, asset)

# Combine individual rolling metrics plots into a single image
metrics_images = [Image.open(os.path.join(output_folder,␣
 ↪f"{asset}_rolling_metrics.png")) for asset in data.columns]
width, height = metrics_images[0].size
combined_metrics_image = Image.new('RGB', (width * 2, height *␣
 ↪((len(metrics_images) + 1) // 2)), (255, 255, 255))

for idx, image in enumerate(metrics_images):
    x_offset = (idx % 2) * width
    y_offset = (idx // 2) * height
    combined_metrics_image.paste(image, (x_offset, y_offset))

combined_metrics_image_path = os.path.join(output_folder,␣
 ↪"combined_rolling_metrics.png")
combined_metrics_image.save(combined_metrics_image_path, format='PNG')
display(combined_metrics_image)
```

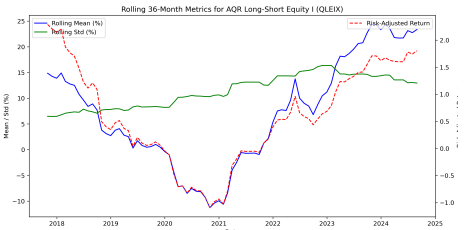


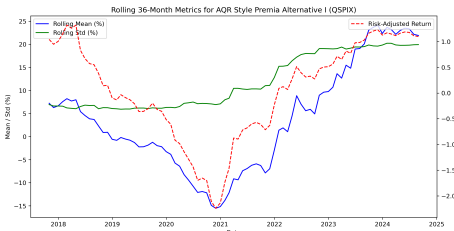


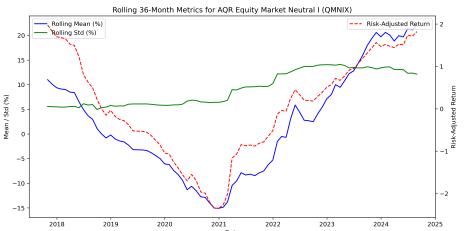


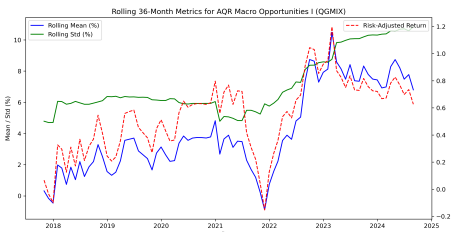


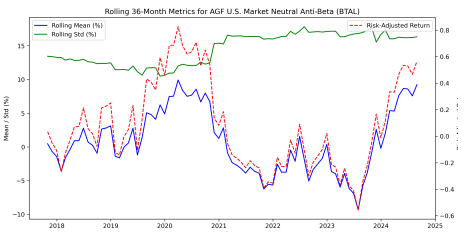


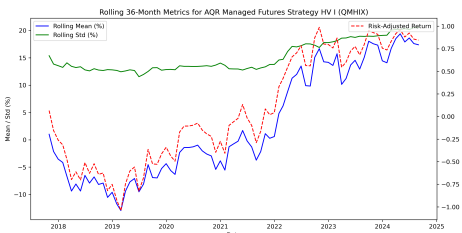


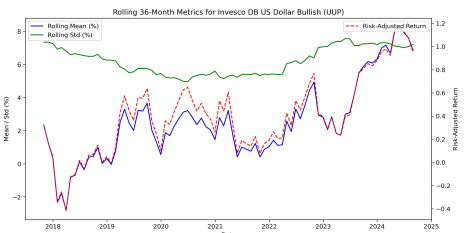


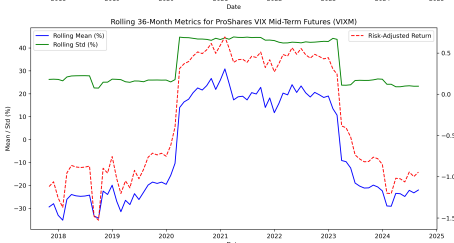












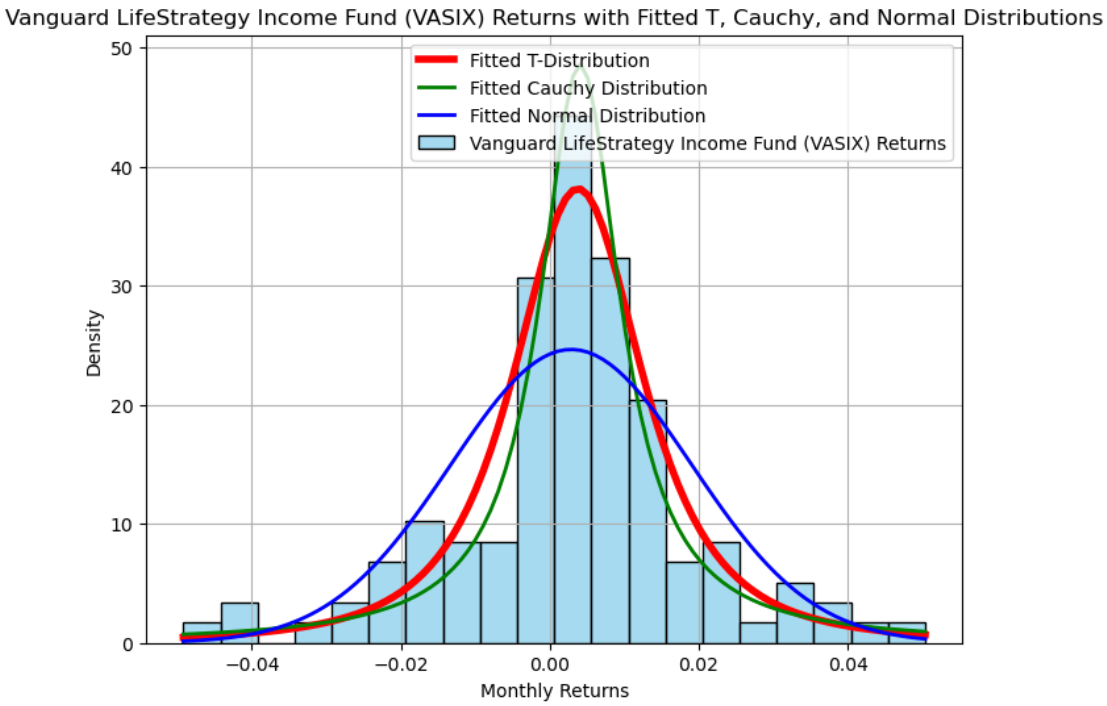
Appendix B1 - Ian Exploratory Data Analysis

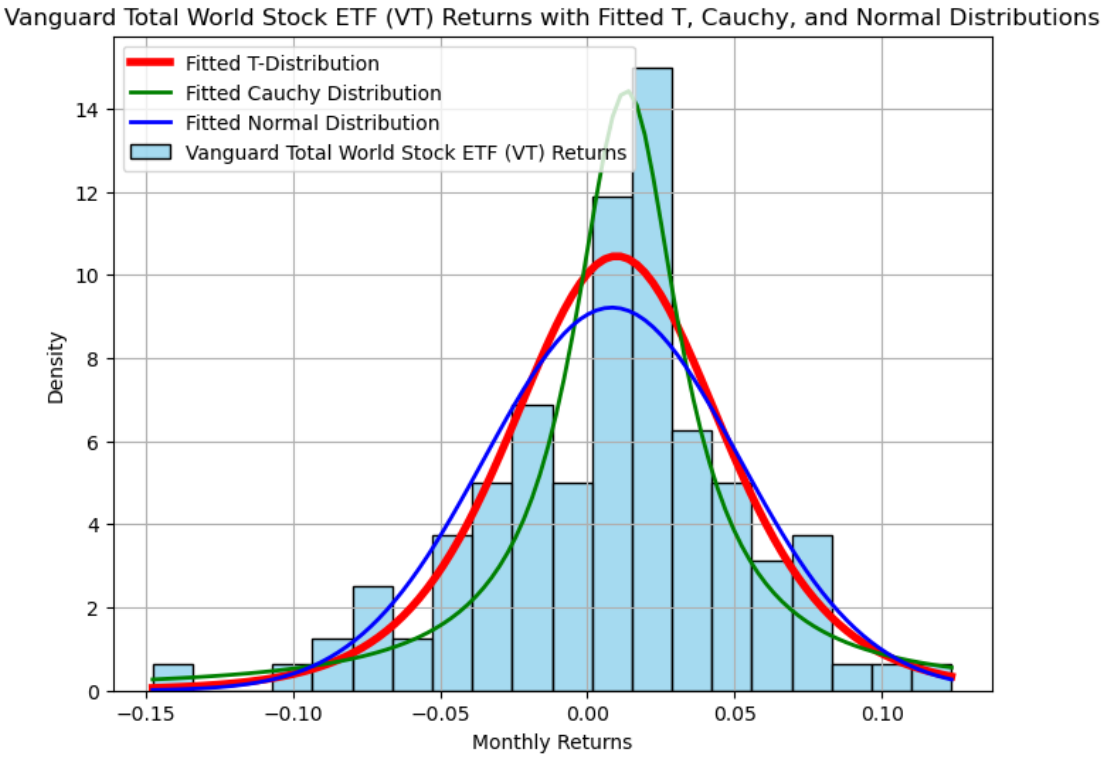
15 Raw Return Histograms with Fitted Probability Distributions (Figure 8)

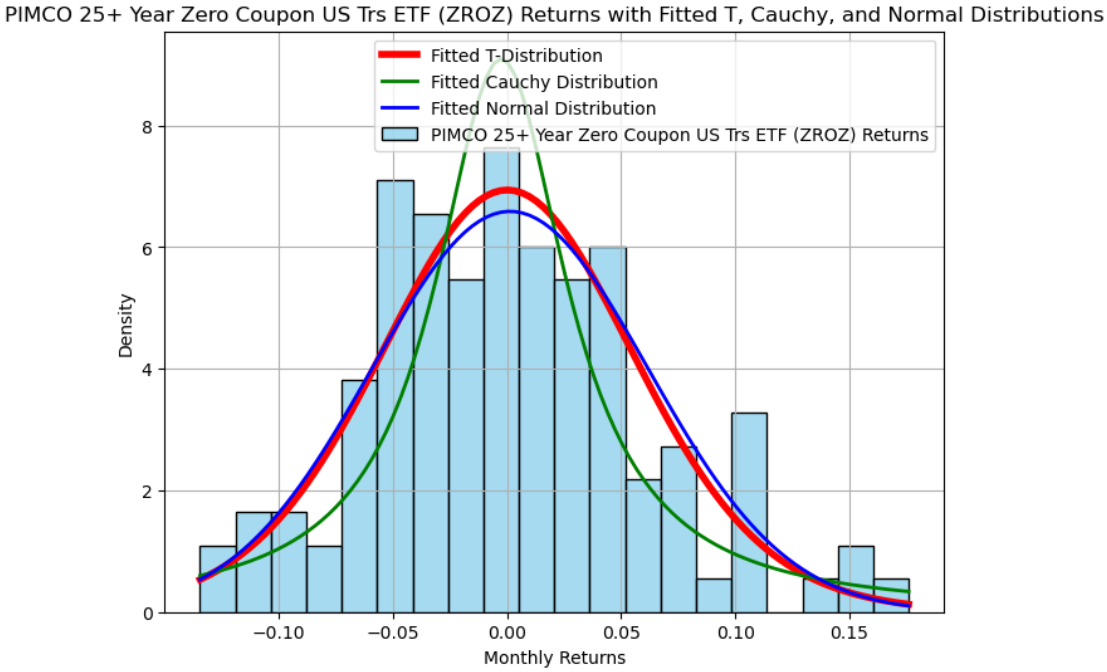
[18]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import t, cauchy, norm

# Comment-based snippet name for the folder
snippet_name = "RAW_DATA_HISTOGRAMS_WITH_FITTED_PROBABILITY_DISTRIBUTIONS"

# Create the directory to save plots
output_folder = snippet_name
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# Load the dataset to inspect the contents
file_path = 'Opportunity_Set.xlsx'
data = pd.read_excel(file_path)

# Create a list to hold individual figures data
figures_data = []

# Generate histogram plots with fitted distributions for each asset (excluding␣
 ↪the 'Date' index)
for asset in data.columns:
    if asset != 'Date':
        asset_returns = data[asset].dropna()  # Ensure no NaNs in returns data
        x = np.linspace(asset_returns.min(), asset_returns.max(), 100)

        # Fit the T-distribution
        df_t, loc_t, scale_t = t.fit(asset_returns)
        pdf_t = t.pdf(x, df_t, loc_t, scale_t)

        # Fit the Cauchy distribution
        loc_cauchy, scale_cauchy = cauchy.fit(asset_returns)
        pdf_cauchy = cauchy.pdf(x, loc_cauchy, scale_cauchy)

        # Fit the Normal distribution
        mu_normal, std_normal = norm.fit(asset_returns)
        pdf_normal = norm.pdf(x, mu_normal, std_normal)
```

```
        # Save the data needed to recreate the plots
        figures_data.append((asset_returns, x, pdf_t, pdf_cauchy, pdf_normal,␣
 ↪asset))

        # Plot the asset returns with fitted distributions
        fig, ax = plt.subplots(figsize=(8, 6))
        sns.histplot(asset_returns, bins=20, kde=False, stat='density',␣
 ↪color='skyblue', label=f'{asset} Returns', ax=ax)
        ax.plot(x, pdf_t, 'r-', lw=4, label='Fitted T-Distribution')
        ax.plot(x, pdf_cauchy, 'g-', lw=2, label='Fitted Cauchy Distribution')
        ax.plot(x, pdf_normal, 'b-', lw=2, label='Fitted Normal Distribution')
        ax.set_title(f"{asset} Returns with Fitted T, Cauchy, and Normal␣
 ↪Distributions")
        ax.set_xlabel("Monthly Returns")
        ax.set_ylabel("Density")
        ax.legend()
        ax.grid(True)

        # Save the plot as a PNG file inside the created folder
        fig.savefig(os.path.join(output_folder, f"{asset}_fitted_distributions.
 ↪png"), bbox_inches='tight', dpi=300)  # Save with high resolution

        # Display the plot
        plt.show()

        # Close the figure to save memory
        plt.close(fig)

# Create a new figure to combine all plots into a single image
combined_fig, combined_axes = plt.subplots(len(figures_data) // 2 +␣
 ↪len(figures_data) % 2, 2, figsize=(16, 22))  # Increase figure size for␣
 ↪better readability
combined_axes = combined_axes.flatten()

# Add each individual figure to the combined figure without distortion
for i, (asset_returns, x, pdf_t, pdf_cauchy, pdf_normal, asset) in␣
 ↪enumerate(figures_data):
    ax = combined_axes[i]
    sns.histplot(asset_returns, bins=20, kde=False, stat='density',␣
 ↪color='skyblue', label=f'{asset} Returns', ax=ax)
    ax.plot(x, pdf_t, 'r-', lw=4, label='Fitted T-Distribution')
    ax.plot(x, pdf_cauchy, 'g-', lw=2, label='Fitted Cauchy Distribution')
    ax.plot(x, pdf_normal, 'b-', lw=2, label='Fitted Normal Distribution')
    ax.set_title(f"{asset} Returns with Fitted T, Cauchy, and Normal␣
 ↪Distributions", fontsize=10)
```

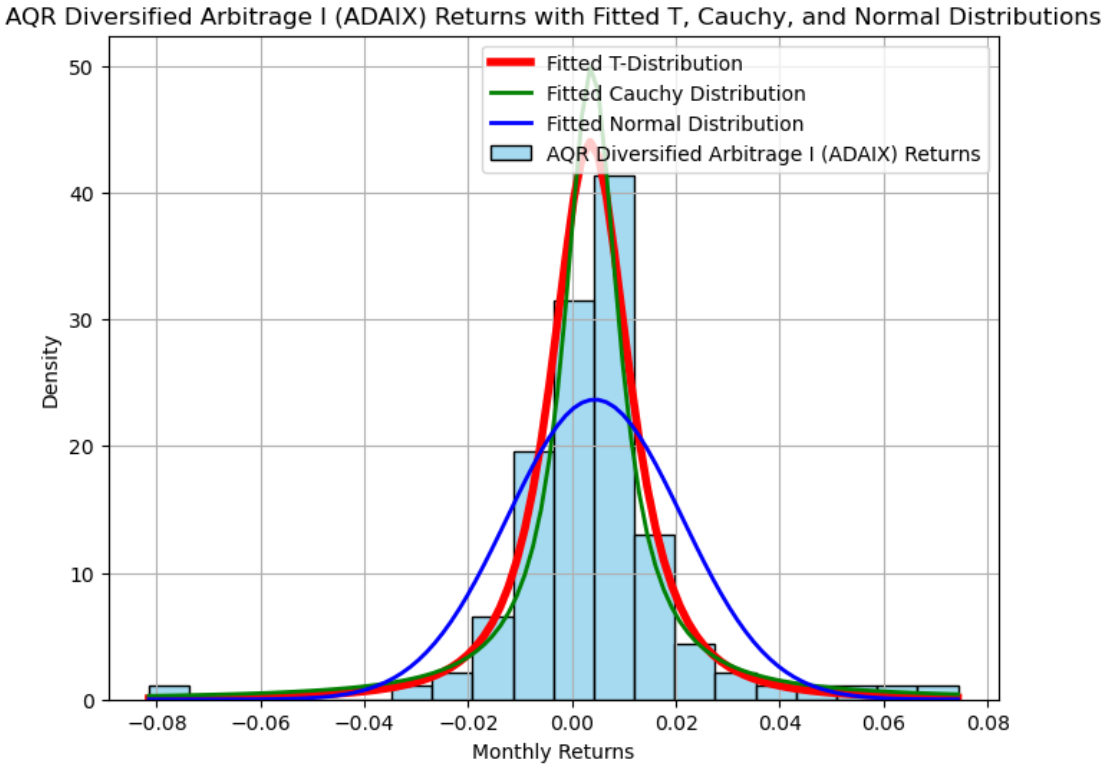
```
    ax.set_xlabel("Monthly Returns", fontsize=8)
    ax.set_ylabel("Density", fontsize=8)
    ax.tick_params(axis='x', rotation=45, labelsize=8)
    ax.tick_params(axis='y', labelsize=8)
    ax.grid(True)
    ax.legend(fontsize=8)

# Hide any unused subplots
for j in range(len(figures_data), len(combined_axes)):
    combined_fig.delaxes(combined_axes[j])

# Adjust layout to prevent overlap
combined_fig.tight_layout()

# Save the combined figure as a PNG file
combined_fig.savefig(os.path.join(output_folder, "combined_fitted_distributions.
 ↪png"), bbox_inches='tight', dpi=300)  # Save with high resolution

# Display the combined plot in Jupyter Notebook
plt.show()
```

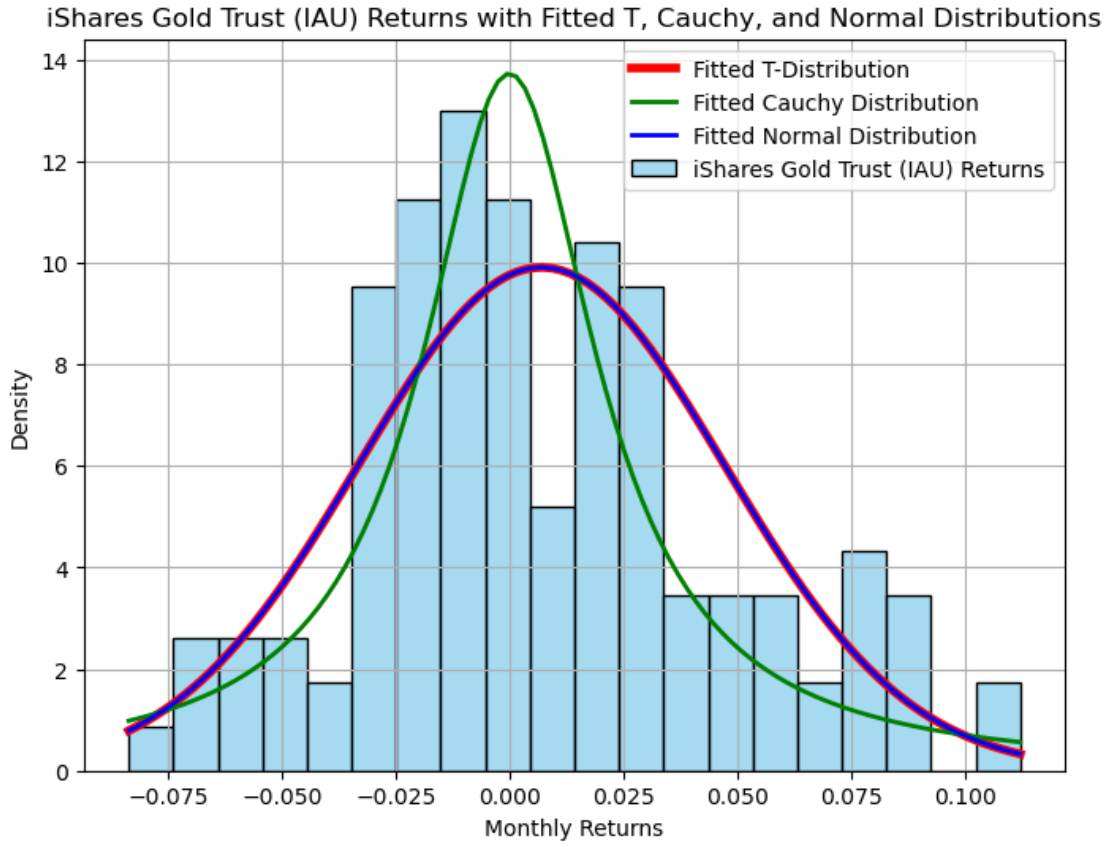




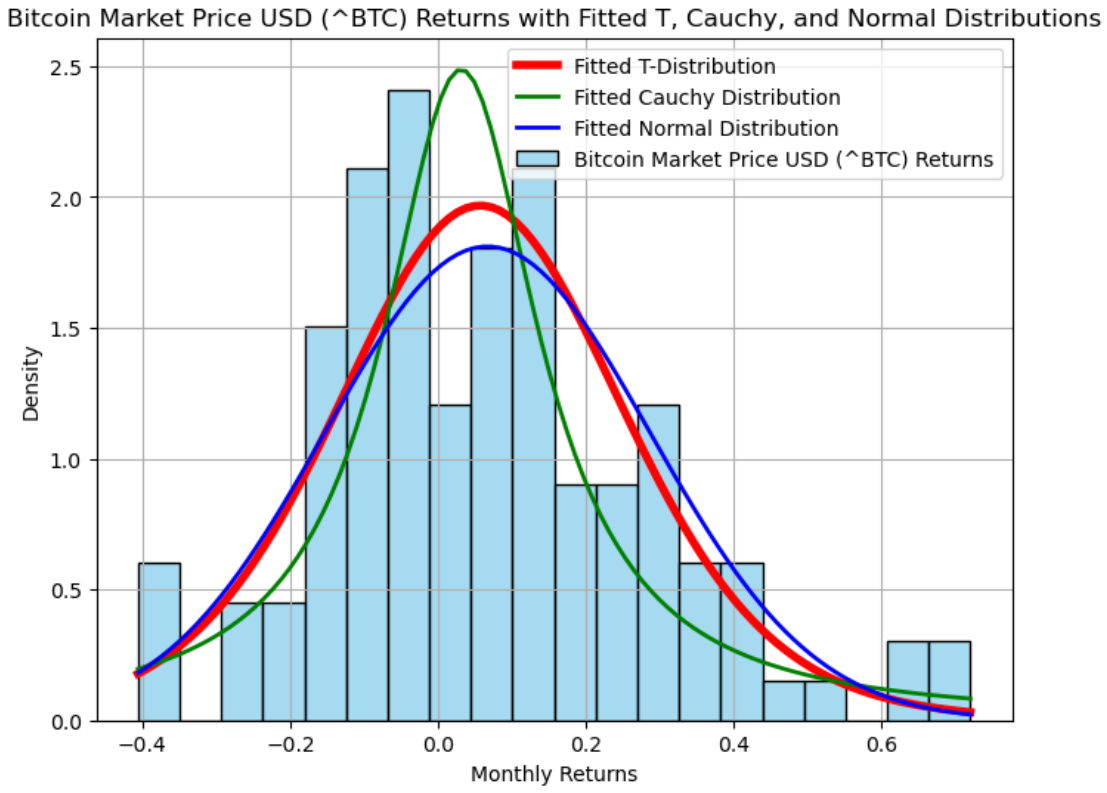


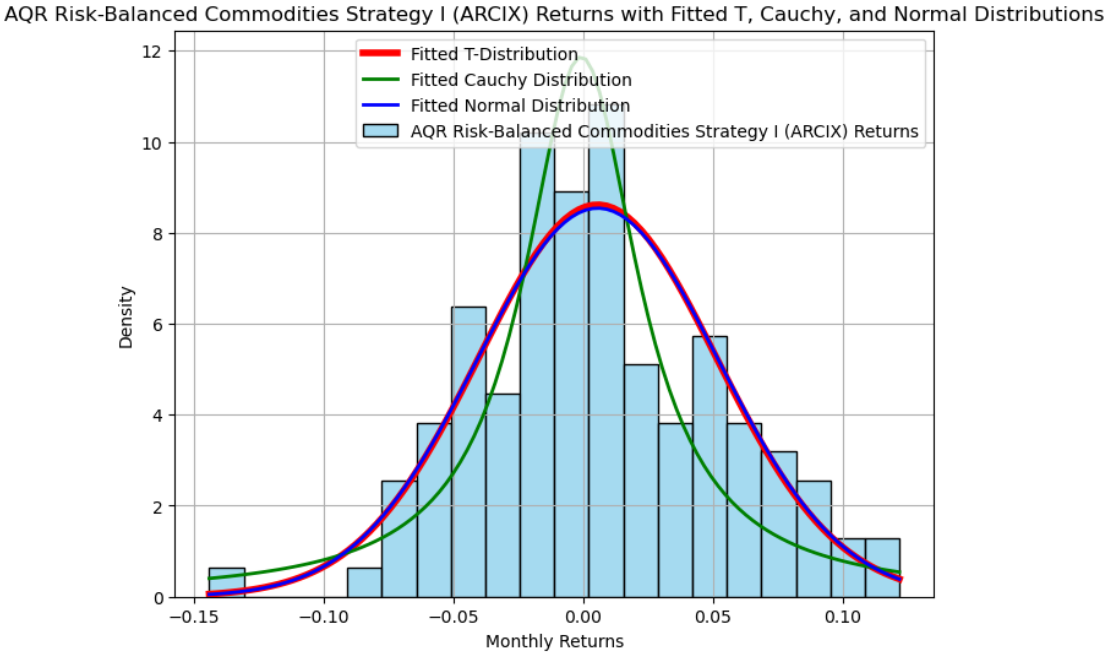
Appendix B1 - Ian Exploratory Data Analysis



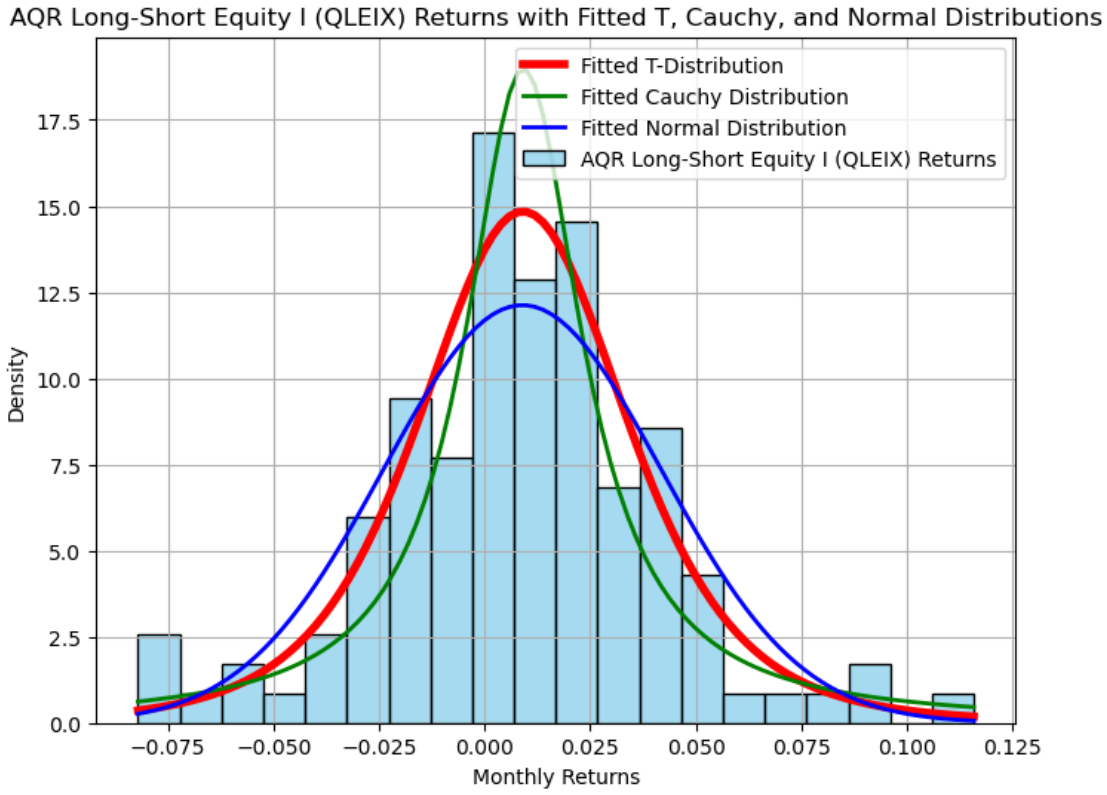


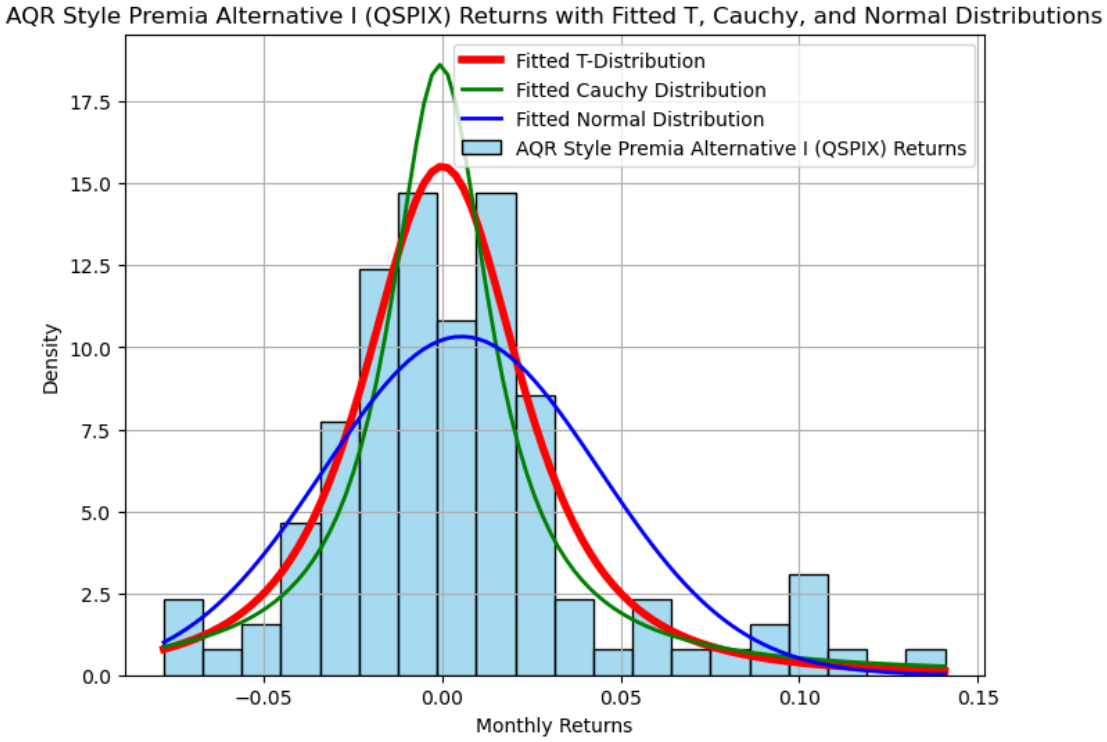
Appendix B1 - Ian Exploratory Data Analysis



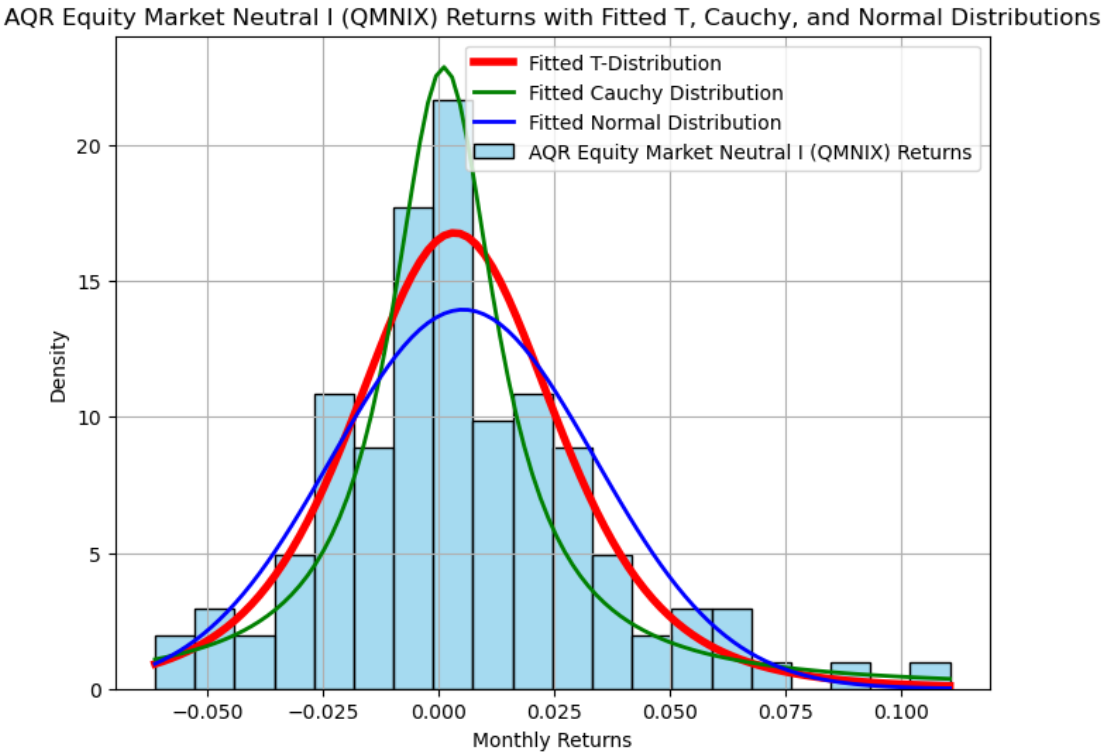


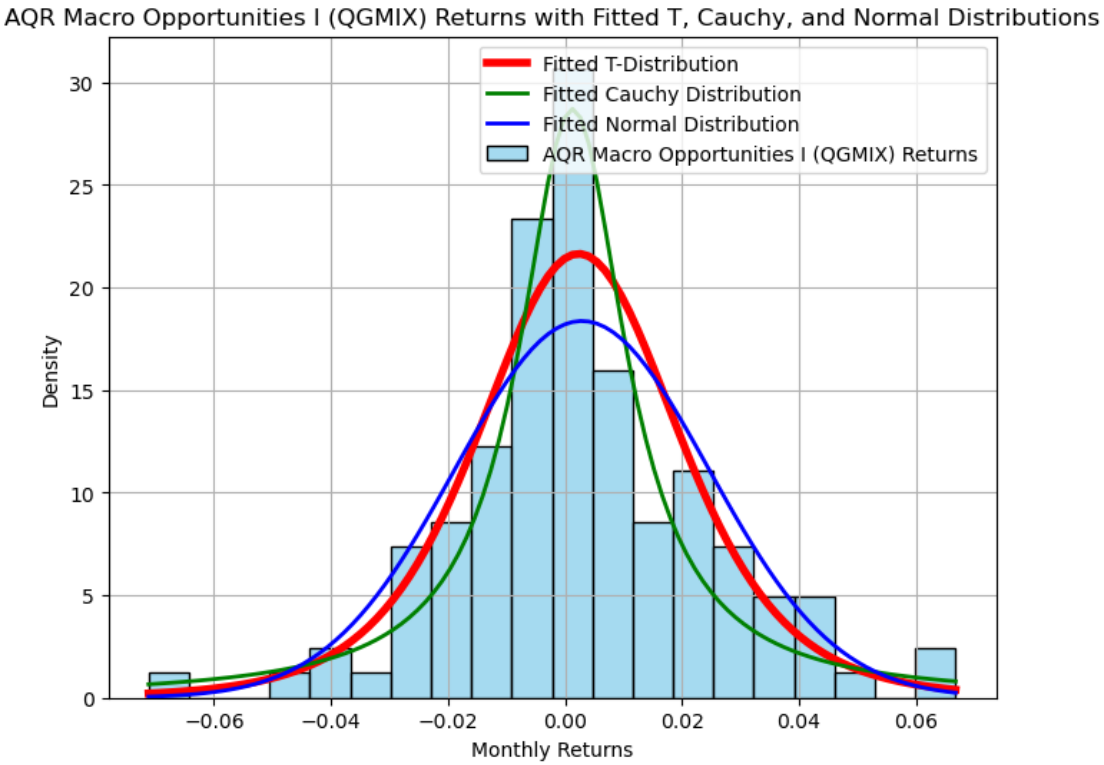
Appendix B1 - Ian Exploratory Data Analysis



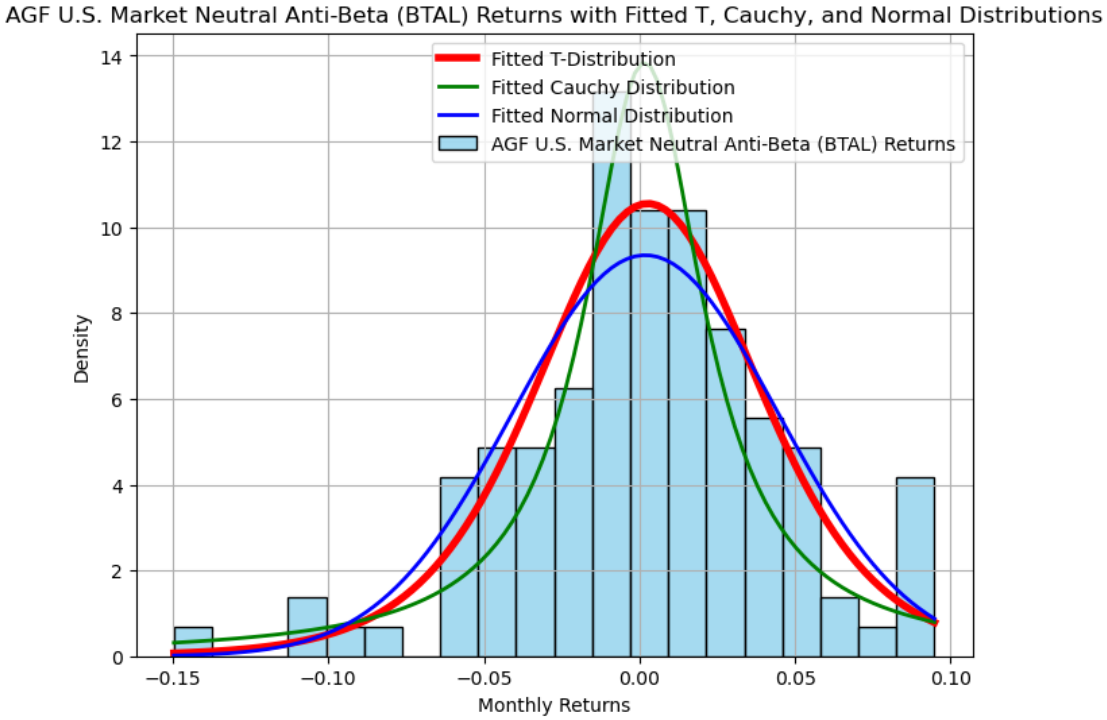


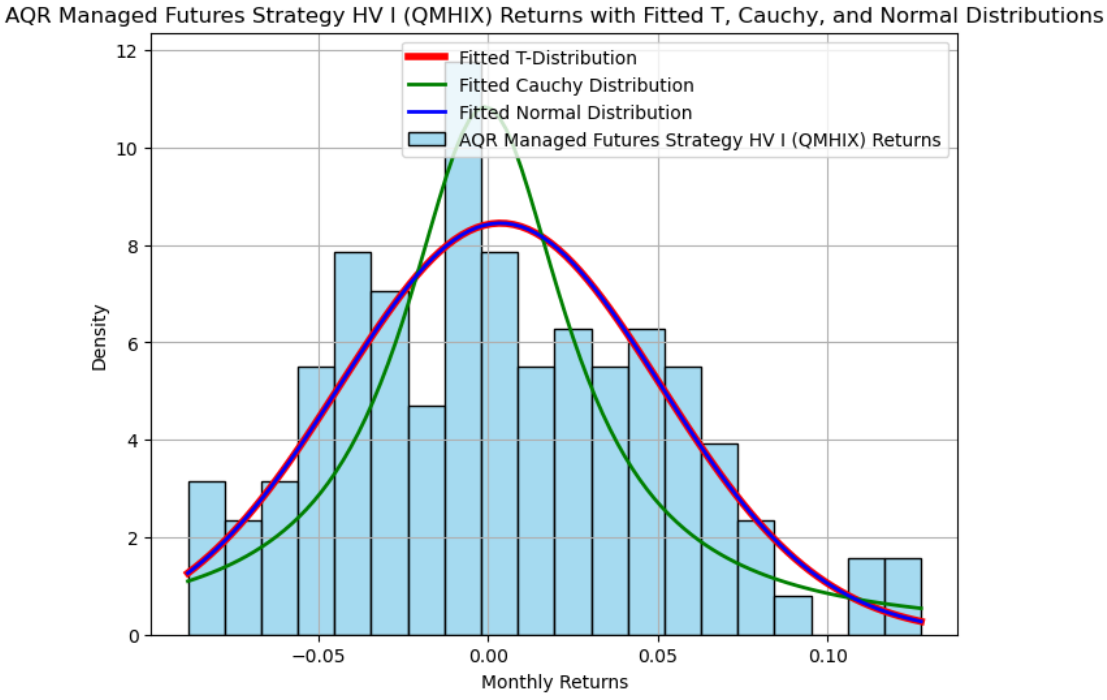
Appendix B1 - Ian Exploratory Data Analysis



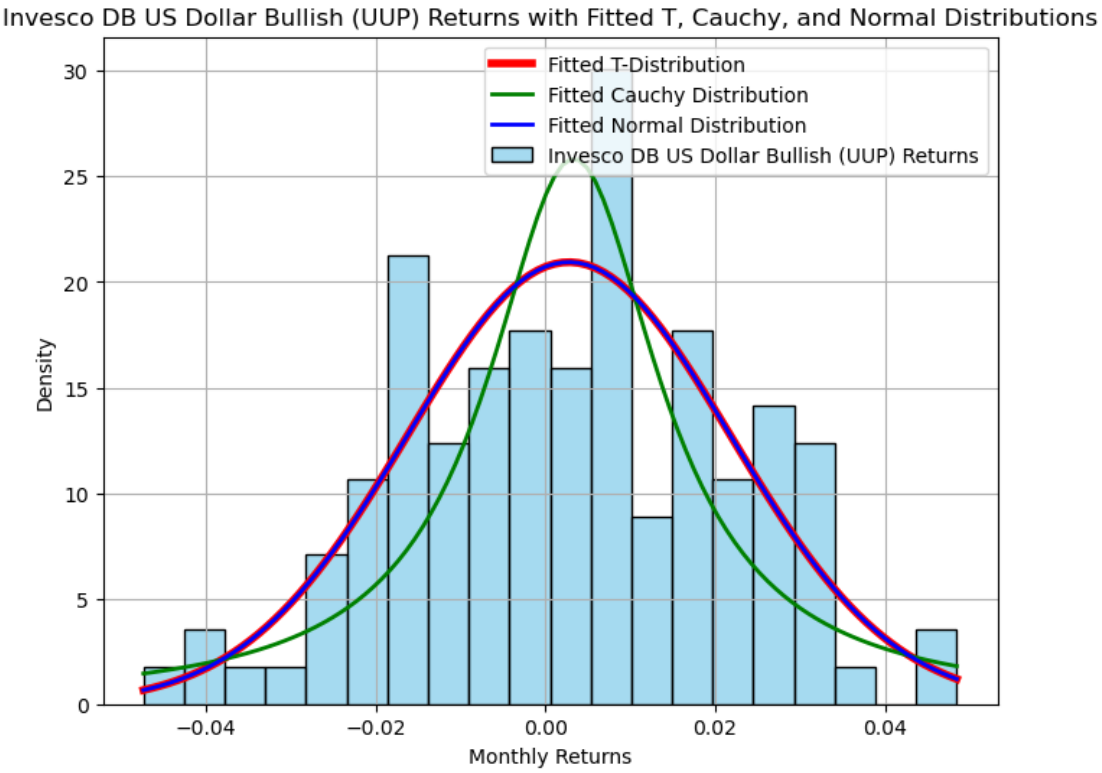


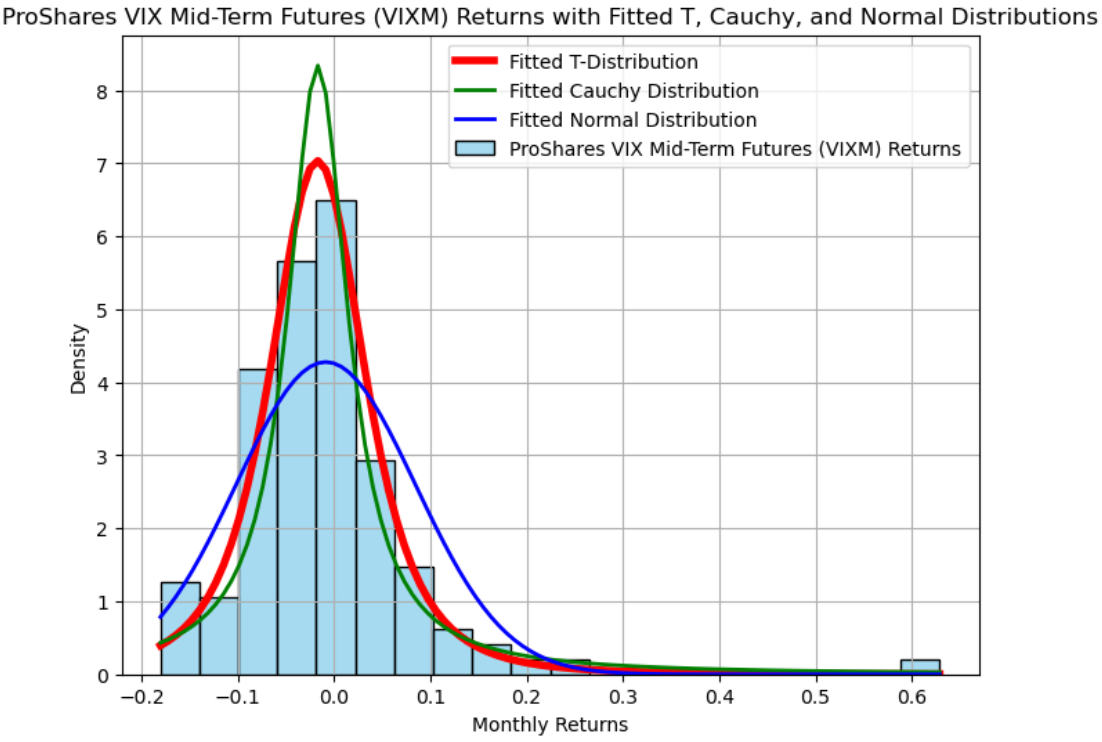
Appendix B1 - Ian Exploratory Data Analysis



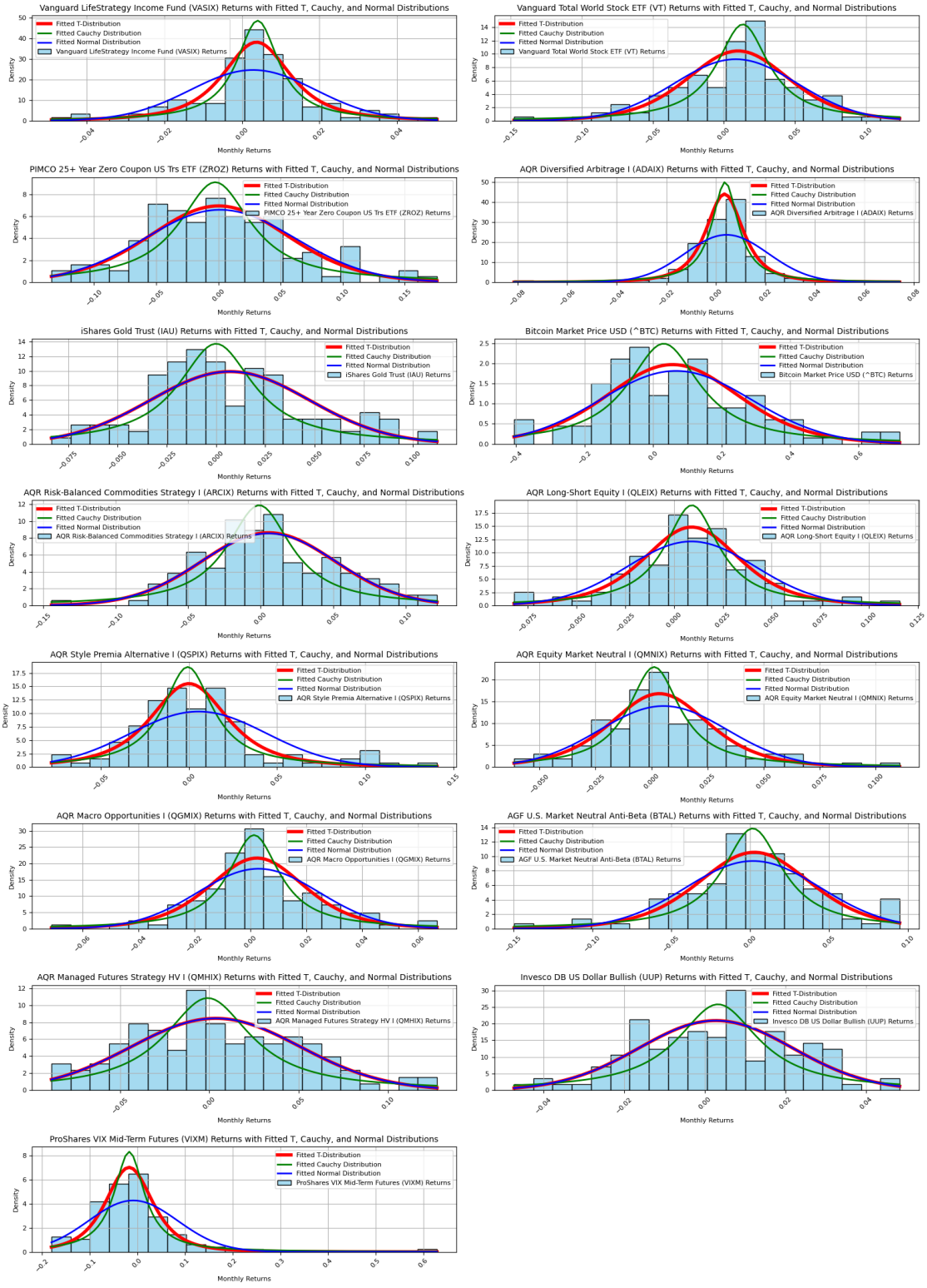


Appendix B1 - Ian Exploratory Data Analysis





Appendix B1 - Ian Exploratory Data Analysis



Appendix B1 - Ian Exploratory Data Analysis

16 Non-Parametric Bootstrapped Return Histograms with Fitted Probability Distributions (10,000 iterations) (Not an exhibit in the report)

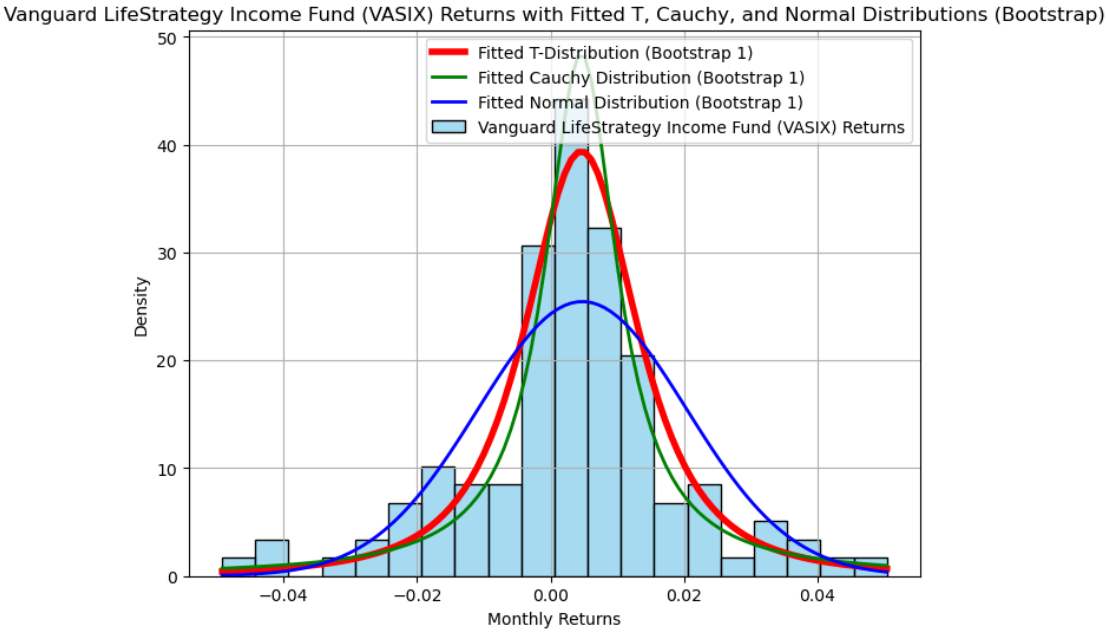
[23]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import t, cauchy, norm
import os

# Comment-based snippet name for the folder
snippet_name = "BOOTSTRAP_FITTED_DISTRIBUTIONS"

# Create the directory to save plots
output_folder = snippet_name
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# Number of bootstrap iterations
n_iterations = 10000

# Create a list to hold individual figures data
figures_data = []

# Iterate over each asset, excluding the 'Date' column
for asset in data.columns:
    if asset != 'Date':
        print(f"Processing asset: {asset}")  # Debug: Check asset being␣
 ↪processed
        asset_returns = data[asset].dropna()  # Ensure no NaNs in returns data
        print(f"Sample size for {asset}: {len(asset_returns)}")  # Debug: Check␣
 ↪sample size
        x = np.linspace(asset_returns.min(), asset_returns.max(), 100)

        # Non-parametric bootstrap: Resample with replacement
        bootstrap_samples = np.random.choice(asset_returns,␣
 ↪size=(len(asset_returns), n_iterations), replace=True)
        print(f"Generated {n_iterations} bootstrap samples for {asset}")  #␣
 ↪Debug: Check bootstrap generation

        # Use the first bootstrap sample to fit distributions
        first_bootstrap_sample = bootstrap_samples[:, 0]

        # Fit the T-distribution to the first bootstrap sample
        df_t, loc_t, scale_t = t.fit(first_bootstrap_sample)
```

```
        print(f"T-distribution (bootstrap sample 1) for {asset}: df={df_t},␣
↪loc={loc_t}, scale={scale_t}")  # Debug

        # Fit the Cauchy distribution to the first bootstrap sample
        loc_cauchy, scale_cauchy = cauchy.fit(first_bootstrap_sample)
        print(f"Cauchy distribution (bootstrap sample 1) for {asset}:␣
↪loc={loc_cauchy}, scale={scale_cauchy}")  # Debug

        # Fit the Normal distribution to the first bootstrap sample
        mu_normal, std_normal = norm.fit(first_bootstrap_sample)
        print(f"Normal distribution (bootstrap sample 1) for {asset}:␣
↪mu={mu_normal}, std={std_normal}")  # Debug

        # Save the data needed to recreate the plots
        figures_data.append((asset_returns, x, df_t, loc_t, scale_t,␣
↪loc_cauchy, scale_cauchy, mu_normal, std_normal, asset))

        # Plot the original asset returns with the fitted distributions from␣
↪the first bootstrap sample
        fig, ax = plt.subplots(figsize=(8, 6))
        sns.histplot(asset_returns, bins=20, kde=False, stat='density',␣
↪color='skyblue', label=f'{asset} Returns', ax=ax)

        # Plot the T-distribution fitted from the first bootstrap sample
        pdf_t = t.pdf(x, df_t, loc_t, scale_t)
        ax.plot(x, pdf_t, 'r-', lw=4, label='Fitted T-Distribution (Bootstrap␣
↪1)')

        # Plot the Cauchy distribution fitted from the first bootstrap sample
        pdf_cauchy = cauchy.pdf(x, loc_cauchy, scale_cauchy)
        ax.plot(x, pdf_cauchy, 'g-', lw=2, label='Fitted Cauchy Distribution␣
↪(Bootstrap 1)')

        # Plot the Normal distribution fitted from the first bootstrap sample
        pdf_normal = norm.pdf(x, mu_normal, std_normal)
        ax.plot(x, pdf_normal, 'b-', lw=2, label='Fitted Normal Distribution␣
↪(Bootstrap 1)')

        ax.set_title(f"{asset} Returns with Fitted T, Cauchy, and Normal␣
↪Distributions (Bootstrap)")
        ax.set_xlabel("Monthly Returns")
        ax.set_ylabel("Density")
        ax.legend()
        ax.grid(True)

        # Save the plot as a PNG file inside the created folder
```

```
        fig.savefig(os.path.join(output_folder,␣
 ↪f"{asset}_bootstrap_fitted_distributions.png"), bbox_inches='tight',␣
 ↪dpi=300)  # Save with high resolution

        # Display the plot
        plt.show()

        # Close the figure to save memory
        plt.close(fig)

# Create a new figure to combine all plots into a single image
combined_fig, combined_axes = plt.subplots(len(figures_data) // 2 +␣
 ↪len(figures_data) % 2, 2, figsize=(16, 22))  # Increase figure size for␣
 ↪better readability
combined_axes = combined_axes.flatten()

# Add each individual figure to the combined figure without distortion
for i, (asset_returns, x, df_t, loc_t, scale_t, loc_cauchy, scale_cauchy,␣
 ↪mu_normal, std_normal, asset) in enumerate(figures_data):
    ax = combined_axes[i]
    sns.histplot(asset_returns, bins=20, kde=False, stat='density',␣
 ↪color='skyblue', label=f'{asset} Returns', ax=ax)
    pdf_t = t.pdf(x, df_t, loc_t, scale_t)
    ax.plot(x, pdf_t, 'r-', lw=4, label='Fitted T-Distribution (Bootstrap 1)')
    pdf_cauchy = cauchy.pdf(x, loc_cauchy, scale_cauchy)
    ax.plot(x, pdf_cauchy, 'g-', lw=2, label='Fitted Cauchy Distribution␣
 ↪(Bootstrap 1)')
    pdf_normal = norm.pdf(x, mu_normal, std_normal)
    ax.plot(x, pdf_normal, 'b-', lw=2, label='Fitted Normal Distribution␣
 ↪(Bootstrap 1)')
    ax.set_title(f"{asset} Returns with Fitted T, Cauchy, and Normal␣
 ↪Distributions (Bootstrap)", fontsize=10)
    ax.set_xlabel("Monthly Returns", fontsize=8)
    ax.set_ylabel("Density", fontsize=8)
    ax.tick_params(axis='x', rotation=45, labelsize=8)
    ax.tick_params(axis='y', labelsize=8)
    ax.grid(True)
    ax.legend(fontsize=8)

# Hide any unused subplots
for j in range(len(figures_data), len(combined_axes)):
    combined_fig.delaxes(combined_axes[j])

# Adjust layout to prevent overlap
combined_fig.tight_layout()
```

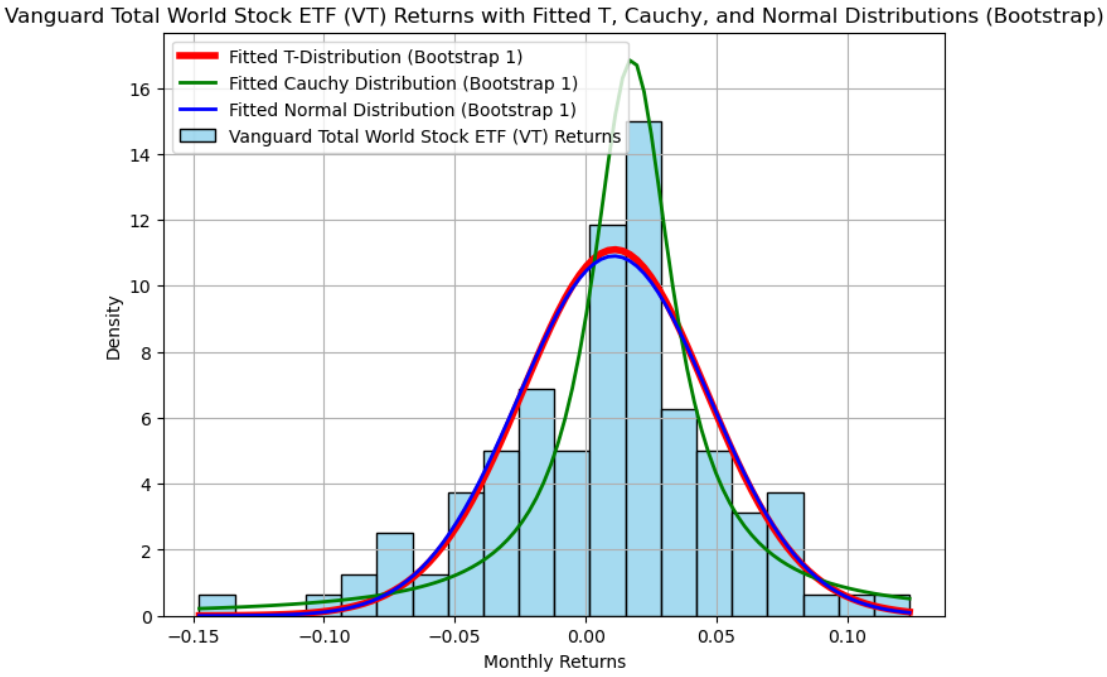
```
# Save the combined figure as a PNG file
combined_fig.savefig(os.path.join(output_folder,␣
 ↪"combined_bootstrap_fitted_distributions.png"), bbox_inches='tight',␣
 ↪dpi=300)  # Save with high resolution

# Display the combined plot in Jupyter Notebook
plt.show()
```

```
Processing asset: Vanguard LifeStrategy Income Fund (VASIX)
Sample size for Vanguard LifeStrategy Income Fund (VASIX): 118
Generated 10000 bootstrap samples for Vanguard LifeStrategy Income Fund (VASIX)
T-distribution (bootstrap sample 1) for Vanguard LifeStrategy Income Fund
(VASIX): df=2.204535572827159, loc=0.004616618338164209,
scale=0.009070137985504132
Cauchy distribution (bootstrap sample 1) for Vanguard LifeStrategy Income Fund
(VASIX): loc=0.004586371841430662, scale=0.006569137954711912
Normal distribution (bootstrap sample 1) for Vanguard LifeStrategy Income Fund
(VASIX): mu=0.004766101694915255, std=0.015673859379682624
```

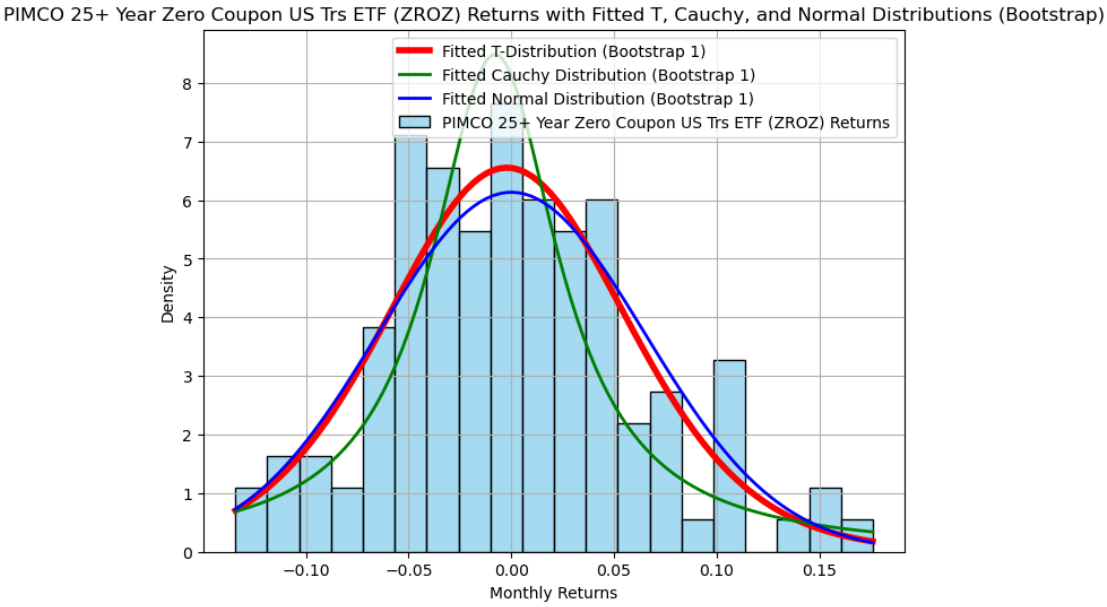



```
Processing asset: Vanguard Total World Stock ETF (VT)
Sample size for Vanguard Total World Stock ETF (VT): 118
Generated 10000 bootstrap samples for Vanguard Total World Stock ETF (VT)
T-distribution (bootstrap sample 1) for Vanguard Total World Stock ETF (VT):
df=42.656054920541166, loc=0.011046908380360803, scale=0.03572720882861005
Cauchy distribution (bootstrap sample 1) for Vanguard Total World Stock ETF
(VT): loc=0.017612798581123353, scale=0.018873235173225388
Normal distribution (bootstrap sample 1) for Vanguard Total World Stock ETF
```

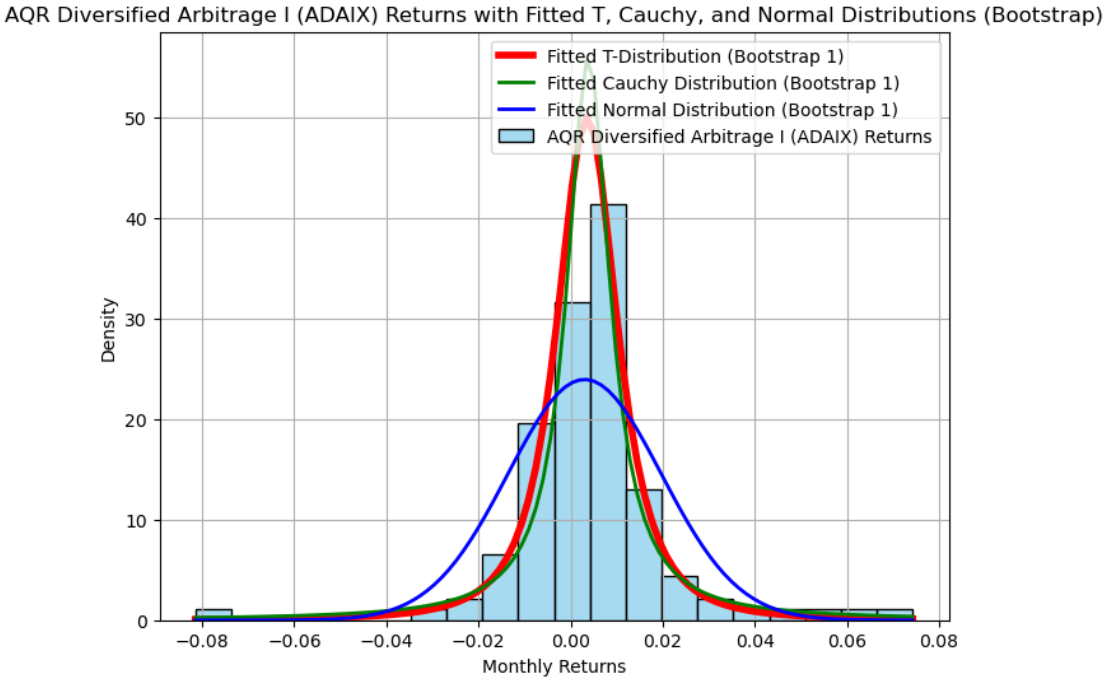
(VT): mu=0.010790677966101694, std=0.036589321836045526



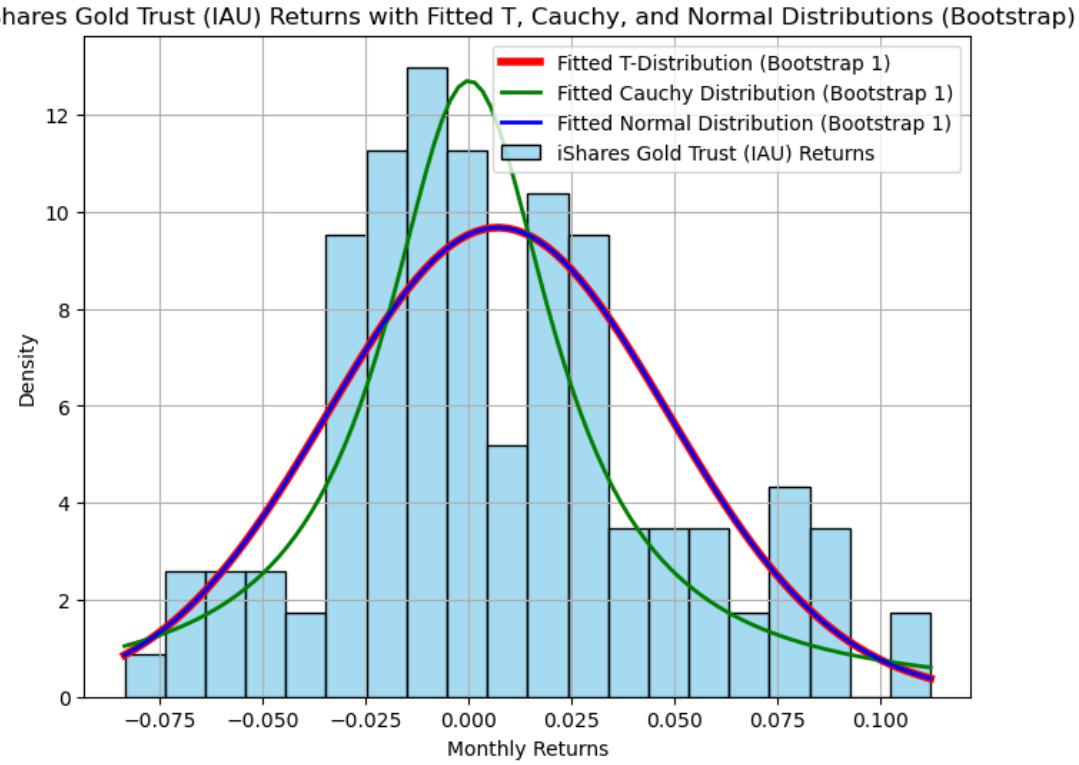
Processing asset: PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)
Sample size for PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ): 118
Generated 10000 bootstrap samples for PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)
T-distribution (bootstrap sample 1) for PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ): df=12.113841337422727, loc=-0.002204202894924164, scale=0.059633785942179704
Cauchy distribution (bootstrap sample 1) for PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ): loc=-0.007920411987304685, scale=0.0374851288890839
Normal distribution (bootstrap sample 1) for PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ): mu=5.3389830508475224e-05, std=0.06503610298825475



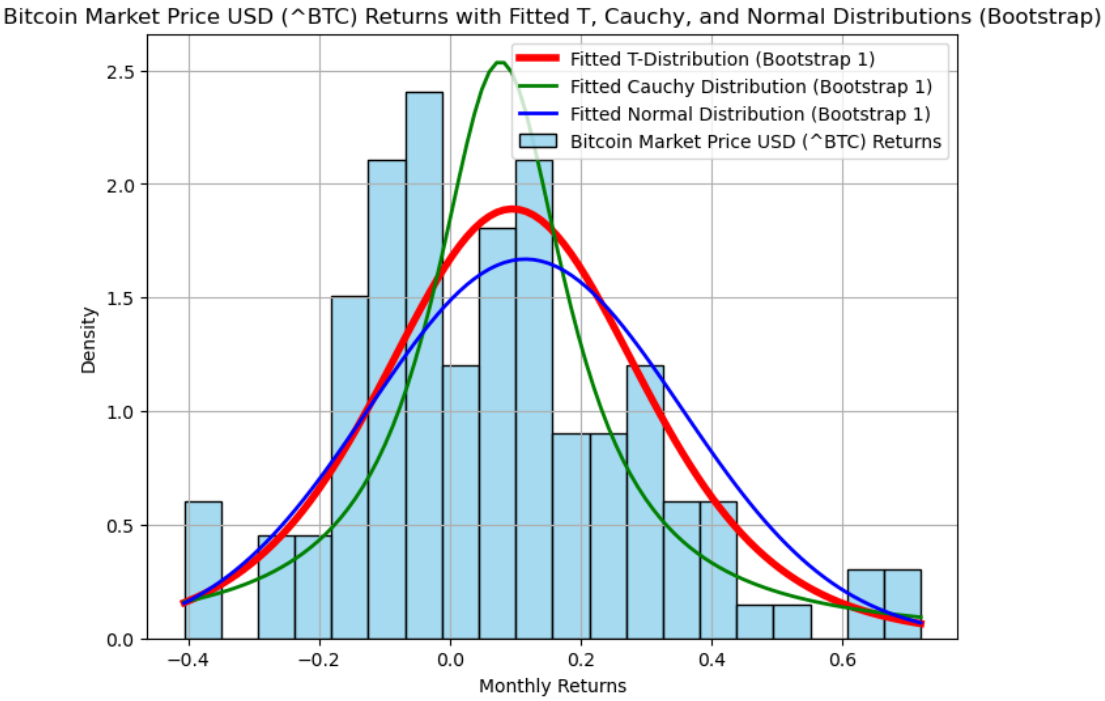

```
Processing asset: AQR Diversified Arbitrage I (ADAIX)
Sample size for AQR Diversified Arbitrage I (ADAIX): 118
Generated 10000 bootstrap samples for AQR Diversified Arbitrage I (ADAIX)
T-distribution (bootstrap sample 1) for AQR Diversified Arbitrage I (ADAIX):
df=2.154571119408492, loc=0.003589952980423613, scale=0.007144667060938941
Cauchy distribution (bootstrap sample 1) for AQR Diversified Arbitrage I
(ADAIX): loc=0.0038814226913452144, scale=0.005690212192535398
Normal distribution (bootstrap sample 1) for AQR Diversified Arbitrage I
(ADAIX): mu=0.003046610169491526, std=0.016681796036923713
```



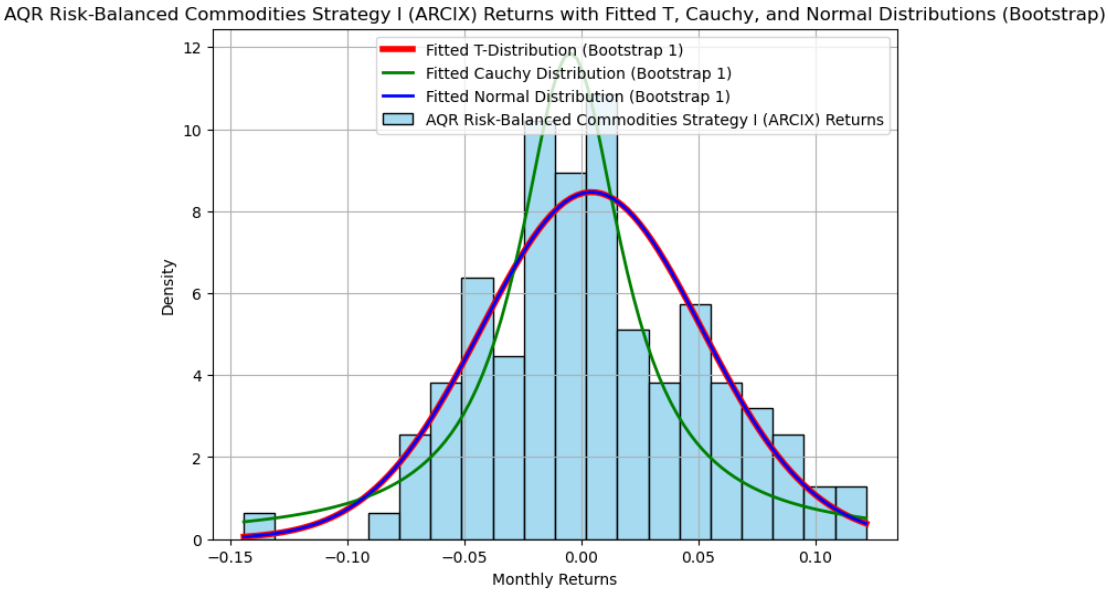

```
Processing asset: iShares Gold Trust (IAU)
Sample size for iShares Gold Trust (IAU): 118
Generated 10000 bootstrap samples for iShares Gold Trust (IAU)
T-distribution (bootstrap sample 1) for iShares Gold Trust (IAU):
df=19448795099.58921, loc=0.007109307847299377, scale=0.041242099198299756
Cauchy distribution (bootstrap sample 1) for iShares Gold Trust (IAU):
loc=8.176025390624736e-05, scale=0.025059235839843777
Normal distribution (bootstrap sample 1) for iShares Gold Trust (IAU):
mu=0.007109322033898304, std=0.04124214149548472
```



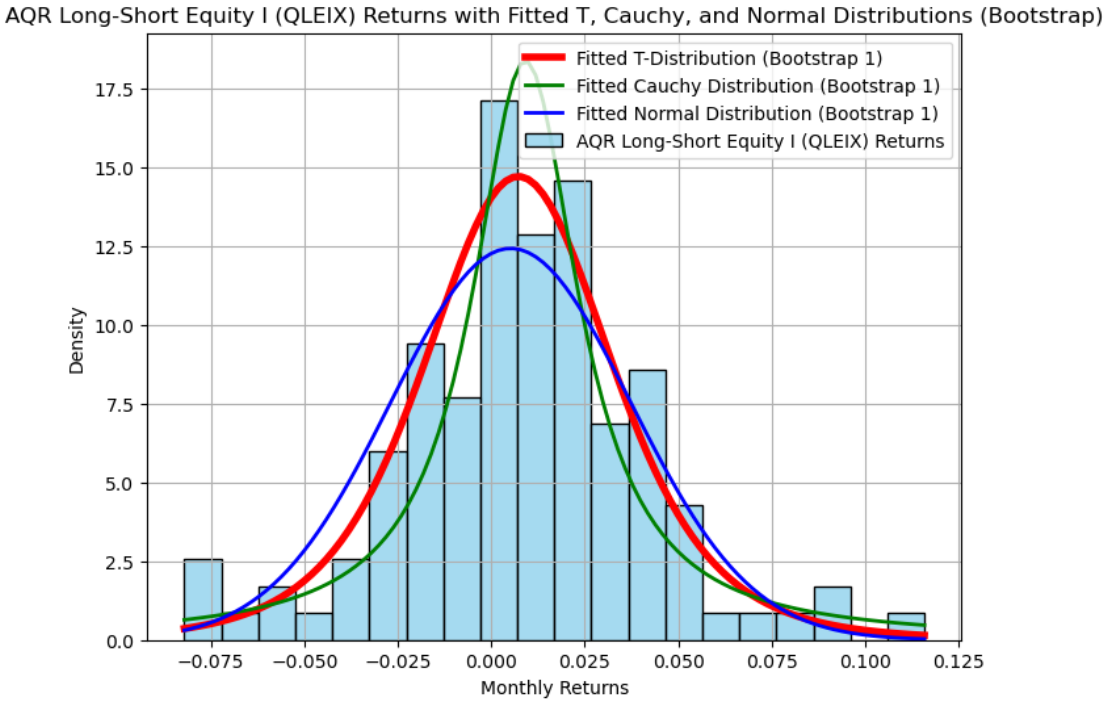

```
Processing asset: Bitcoin Market Price USD (^BTC)
Sample size for Bitcoin Market Price USD (^BTC): 118
Generated 10000 bootstrap samples for Bitcoin Market Price USD (^BTC)
T-distribution (bootstrap sample 1) for Bitcoin Market Price USD (^BTC):
df=6.691724545830047, loc=0.09575876103555714, scale=0.20336266348936424
Cauchy distribution (bootstrap sample 1) for Bitcoin Market Price USD (^BTC):
loc=0.07729415461971556, scale=0.1253720332030407
Normal distribution (bootstrap sample 1) for Bitcoin Market Price USD (^BTC):
mu=0.11542372881355932, std=0.23903781087970946
```



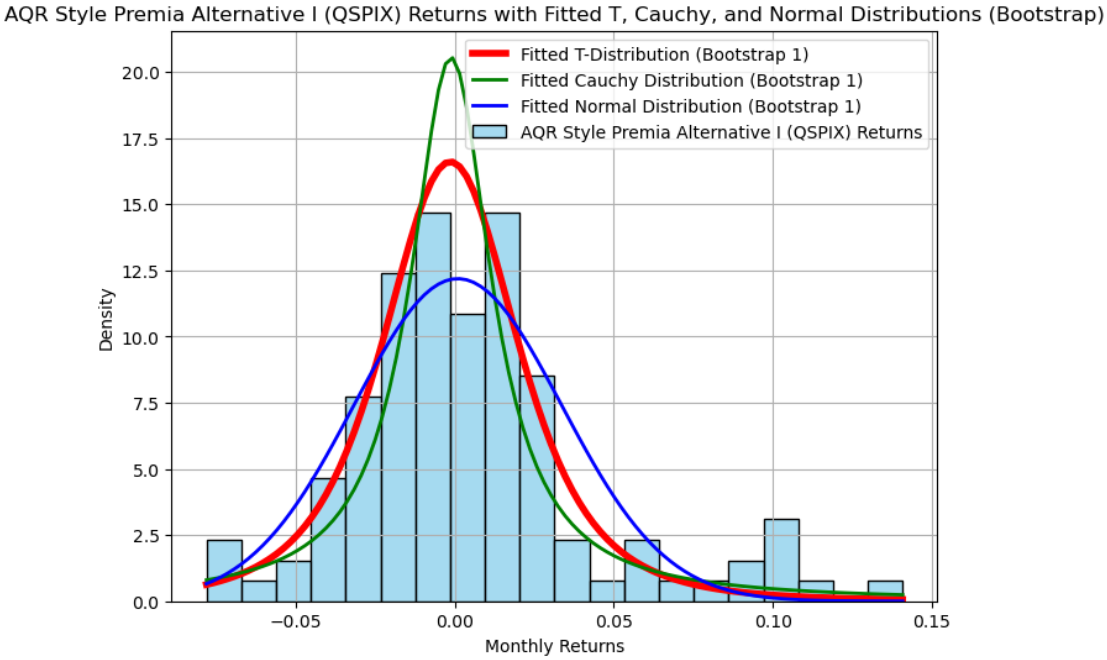

```
Processing asset: AQR Risk-Balanced Commodities Strategy I (ARCIX)
Sample size for AQR Risk-Balanced Commodities Strategy I (ARCIX): 118
Generated 10000 bootstrap samples for AQR Risk-Balanced Commodities Strategy I
(ARCIX)
T-distribution (bootstrap sample 1) for AQR Risk-Balanced Commodities Strategy I
(ARCIX): df=3394474584.0657654, loc=0.0043525126834378074,
scale=0.047128984351194586
Cauchy distribution (bootstrap sample 1) for AQR Risk-Balanced Commodities
Strategy I (ARCIX): loc=-0.00469184761047364, scale=0.026851655731201088
Normal distribution (bootstrap sample 1) for AQR Risk-Balanced Commodities
Strategy I (ARCIX): mu=0.004352542372881354, std=0.04712897865570142
```



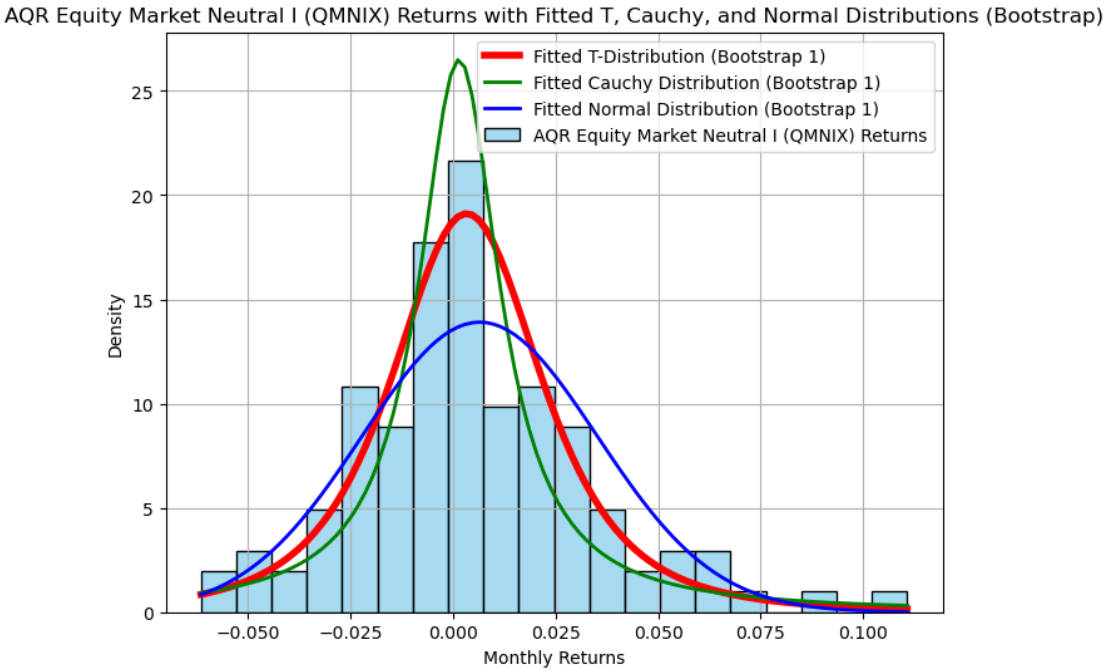

```
Processing asset: AQR Long-Short Equity I (QLEIX)
Sample size for AQR Long-Short Equity I (QLEIX): 118
Generated 10000 bootstrap samples for AQR Long-Short Equity I (QLEIX)
T-distribution (bootstrap sample 1) for AQR Long-Short Equity I (QLEIX):
df=5.252612520881511, loc=0.0072297779725766405, scale=0.025879679335627882
Cauchy distribution (bootstrap sample 1) for AQR Long-Short Equity I (QLEIX):
loc=0.00910498809814453, scale=0.017330052032470702
Normal distribution (bootstrap sample 1) for AQR Long-Short Equity I (QLEIX):
mu=0.005185593220338983, std=0.03211961826034772
```



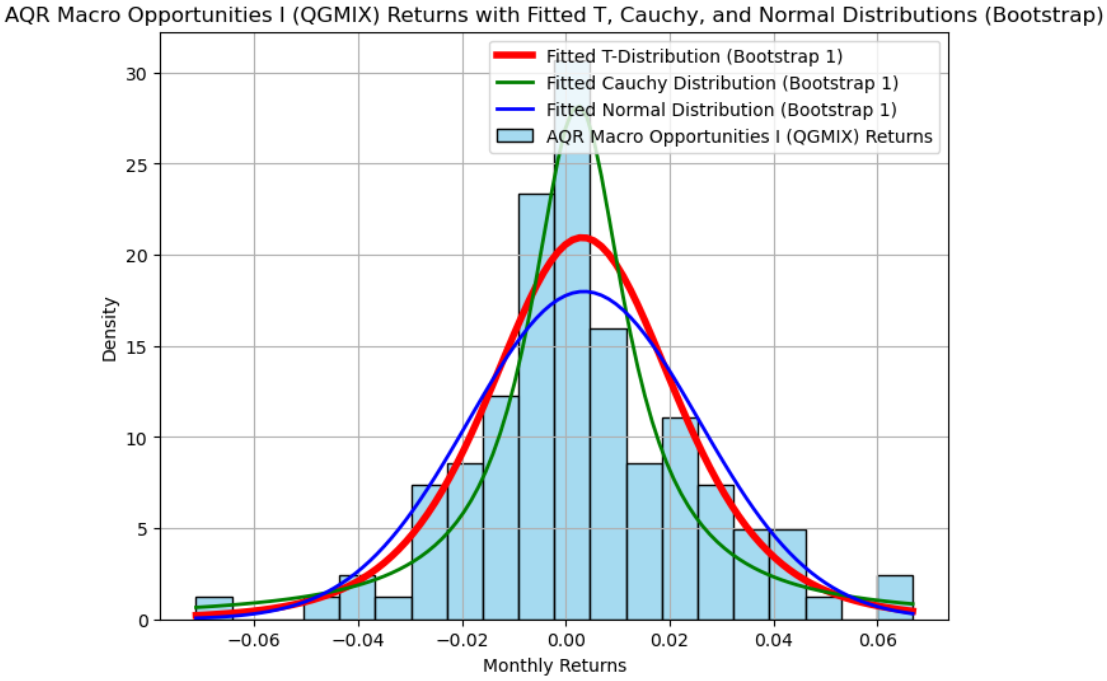

```
Processing asset: AQR Style Premia Alternative I (QSPIX)
Sample size for AQR Style Premia Alternative I (QSPIX): 118
Generated 10000 bootstrap samples for AQR Style Premia Alternative I (QSPIX)
T-distribution (bootstrap sample 1) for AQR Style Premia Alternative I (QSPIX):
df=3.288436286905494, loc=-0.001418549077672707, scale=0.02228012741322024
Cauchy distribution (bootstrap sample 1) for AQR Style Premia Alternative I
(QSPIX): loc=-0.0012144628906250003, scale=0.015482998046875003
Normal distribution (bootstrap sample 1) for AQR Style Premia Alternative I
(QSPIX): mu=0.0009262711864406778, std=0.03272239793740544
```



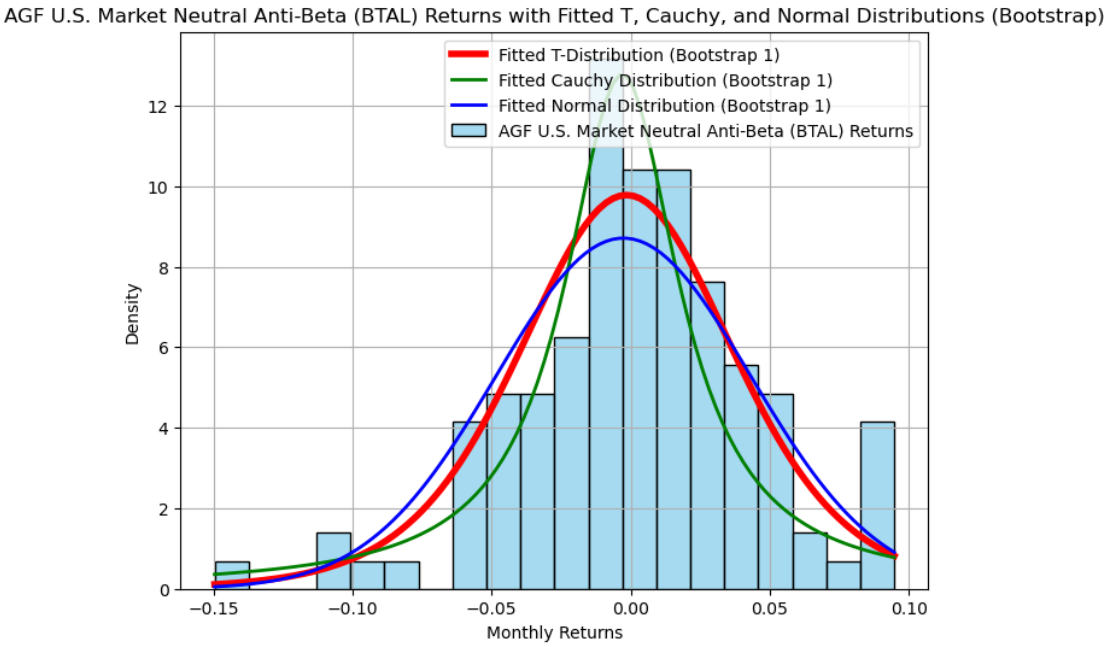

```
Processing asset: AQR Equity Market Neutral I (QMNIX)
Sample size for AQR Equity Market Neutral I (QMNIX): 118
Generated 10000 bootstrap samples for AQR Equity Market Neutral I (QMNIX)
T-distribution (bootstrap sample 1) for AQR Equity Market Neutral I (QMNIX):
df=2.9402221310381007, loc=0.003331117160879359, scale=0.019194384277400903
Cauchy distribution (bootstrap sample 1) for AQR Equity Market Neutral I
(QMNIX): loc=0.0014923535156249998, scale=0.01200941162109375
Normal distribution (bootstrap sample 1) for AQR Equity Market Neutral I
(QMNIX): mu=0.006518644067796611, std=0.028681021744891507
```



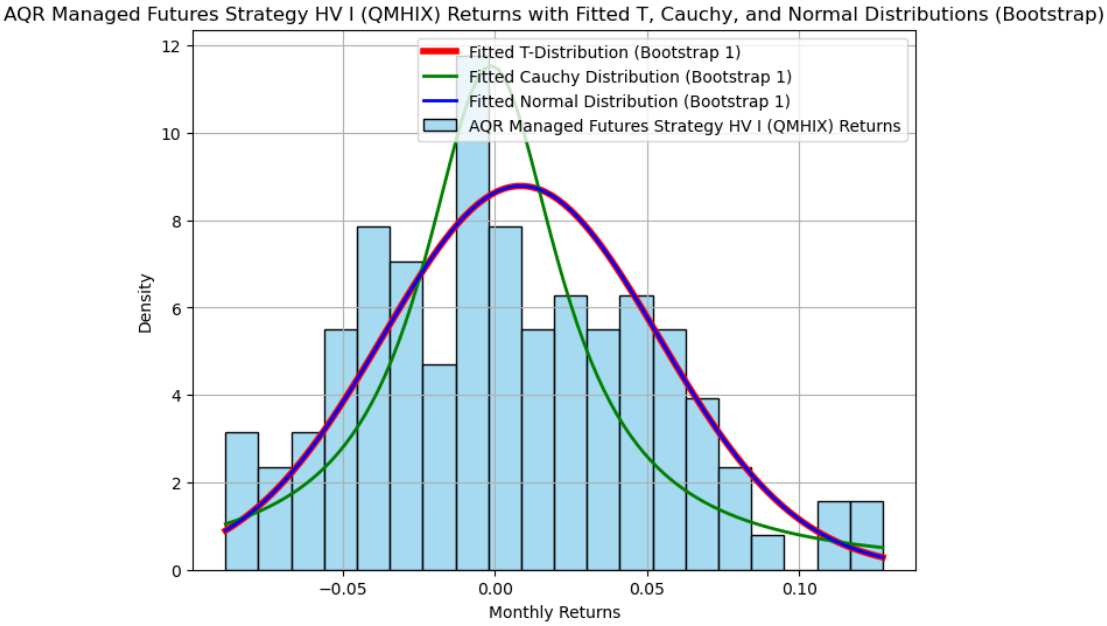

```
Processing asset: AQR Macro Opportunities I (QGMIX)
Sample size for AQR Macro Opportunities I (QGMIX): 118
Generated 10000 bootstrap samples for AQR Macro Opportunities I (QGMIX)
T-distribution (bootstrap sample 1) for AQR Macro Opportunities I (QGMIX):
df=5.854242147327145, loc=0.0031937358652952145, scale=0.018247686062864768
Cauchy distribution (bootstrap sample 1) for AQR Macro Opportunities I (QGMIX):
loc=0.0023553315734863266, scale=0.011294057620763775
Normal distribution (bootstrap sample 1) for AQR Macro Opportunities I (QGMIX):
mu=0.003473728813559322, std=0.022176944158238853
```



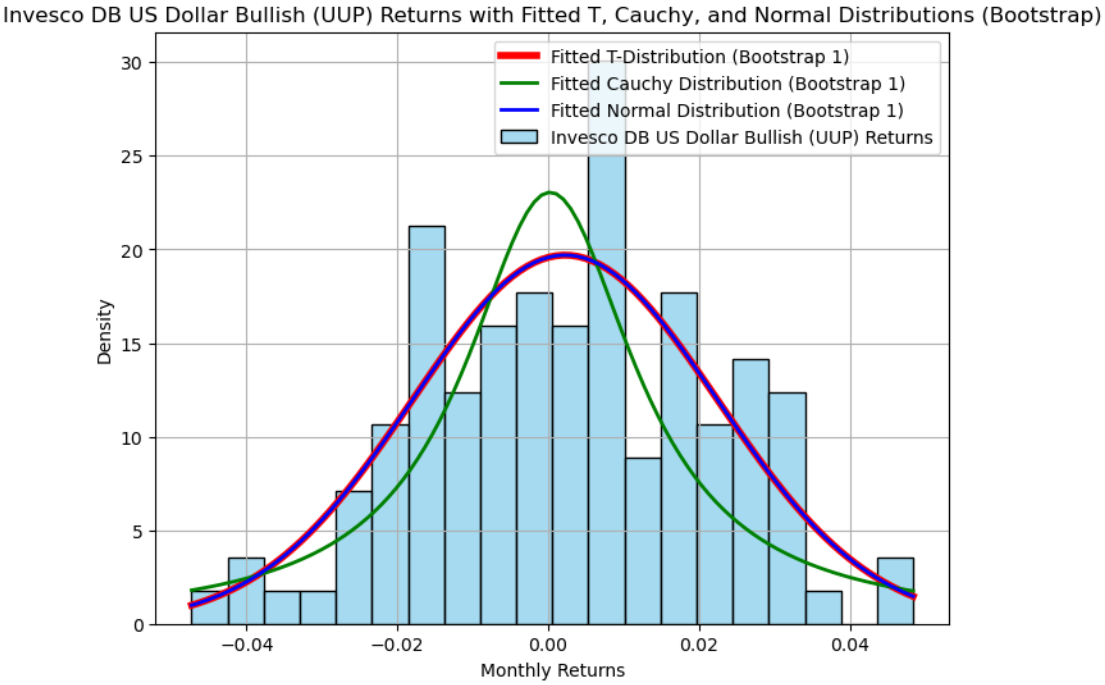

```
Processing asset: AGF U.S. Market Neutral Anti-Beta (BTAL)
Sample size for AGF U.S. Market Neutral Anti-Beta (BTAL): 118
Generated 10000 bootstrap samples for AGF U.S. Market Neutral Anti-Beta (BTAL)
T-distribution (bootstrap sample 1) for AGF U.S. Market Neutral Anti-Beta
(BTAL): df=7.5244837144737815, loc=-0.001474755725036669,
scale=0.03946032031136168
Cauchy distribution (bootstrap sample 1) for AGF U.S. Market Neutral Anti-Beta
(BTAL): loc=-0.0030841846847534196, scale=0.0249155071735382
Normal distribution (bootstrap sample 1) for AGF U.S. Market Neutral Anti-Beta
(BTAL): mu=-0.0026161016949152547, std=0.04579611220701812
```



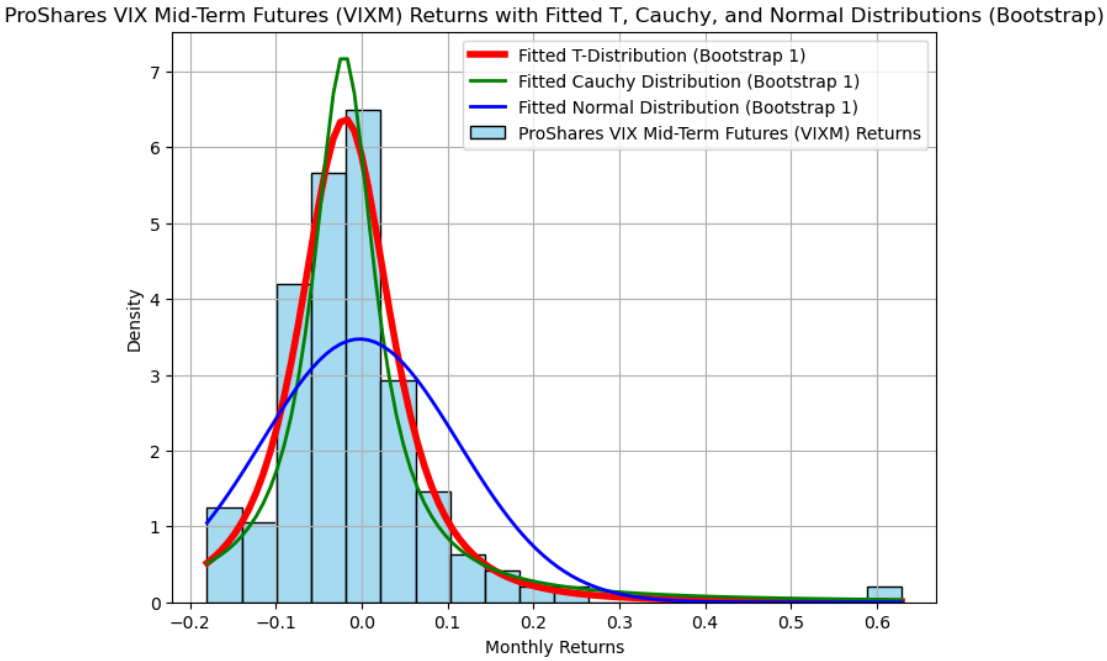

```
Processing asset: AQR Managed Futures Strategy HV I (QMHIX)
Sample size for AQR Managed Futures Strategy HV I (QMHIX): 118
Generated 10000 bootstrap samples for AQR Managed Futures Strategy HV I (QMHIX)
T-distribution (bootstrap sample 1) for AQR Managed Futures Strategy HV I
(QMHIX): df=215425607.84337816, loc=0.008519367155341732,
scale=0.045411743059432755
Cauchy distribution (bootstrap sample 1) for AQR Managed Futures Strategy HV I
(QMHIX): loc=-0.0012177734374999995, scale=0.02759399414062498
Normal distribution (bootstrap sample 1) for AQR Managed Futures Strategy HV I
(QMHIX): mu=0.008519491525423731, std=0.04541182041759346
```

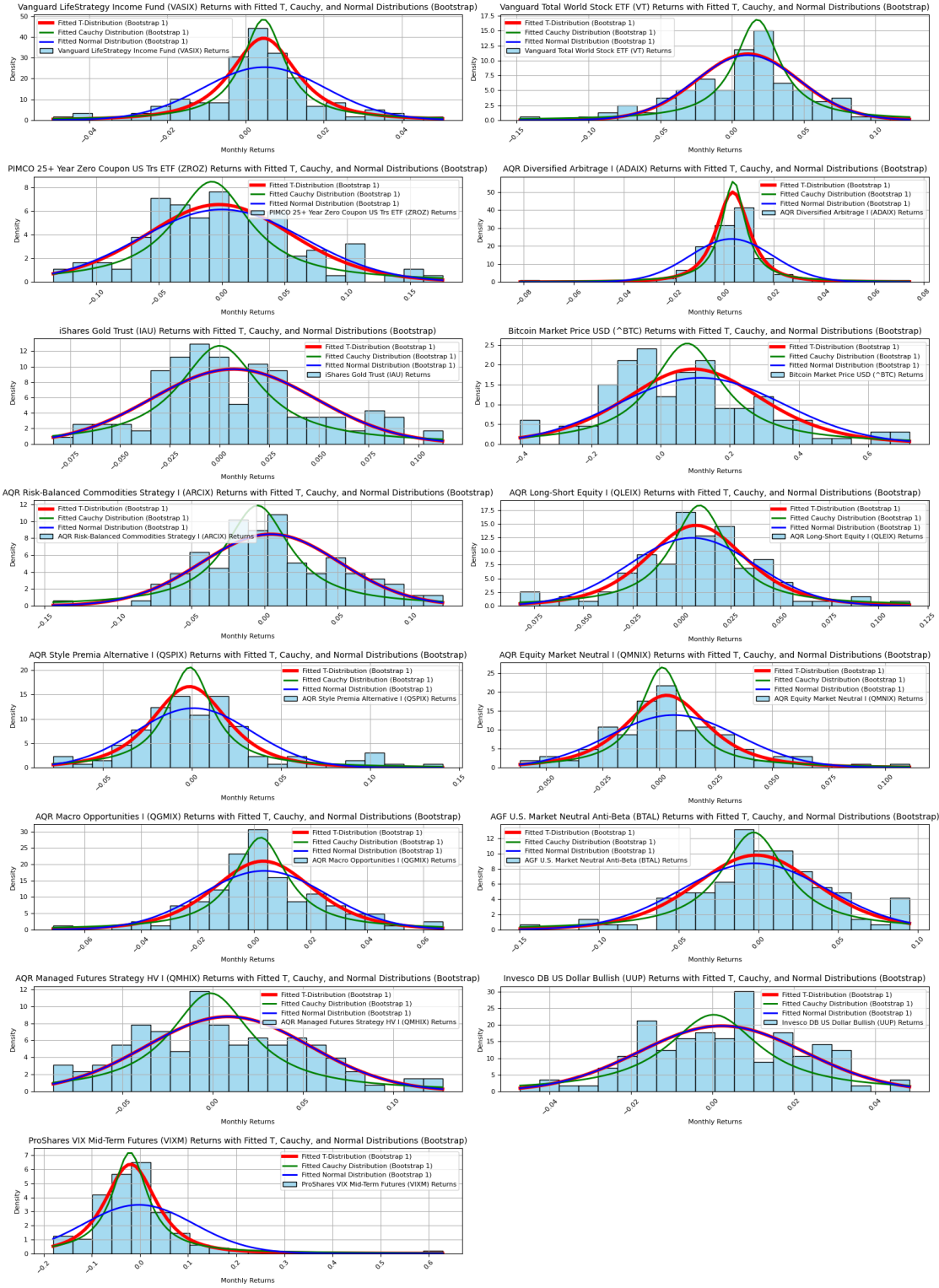



```
Processing asset: Invesco DB US Dollar Bullish (UUP)
Sample size for Invesco DB US Dollar Bullish (UUP): 118
Generated 10000 bootstrap samples for Invesco DB US Dollar Bullish (UUP)
T-distribution (bootstrap sample 1) for Invesco DB US Dollar Bullish (UUP):
df=3170640257.430148, loc=0.0022296703801621444, scale=0.020255244405494803
Cauchy distribution (bootstrap sample 1) for Invesco DB US Dollar Bullish (UUP):
loc=0.0002362935966718836, scale=0.013806712949587869
Normal distribution (bootstrap sample 1) for Invesco DB US Dollar Bullish (UUP):
mu=0.002229661016949152, std=0.02025521196505969
```




```
Processing asset: ProShares VIX Mid-Term Futures (VIXM)
Sample size for ProShares VIX Mid-Term Futures (VIXM): 118
Generated 10000 bootstrap samples for ProShares VIX Mid-Term Futures (VIXM)
T-distribution (bootstrap sample 1) for ProShares VIX Mid-Term Futures (VIXM):
df=2.4489111968240422, loc=-0.019283755577882818, scale=0.05665469905232659
Cauchy distribution (bootstrap sample 1) for ProShares VIX Mid-Term Futures
(VIXM): loc=-0.020733983216285723, scale=0.044046495497226716
Normal distribution (bootstrap sample 1) for ProShares VIX Mid-Term Futures
(VIXM): mu=-0.002135593220338985, std=0.11493853517735418
```

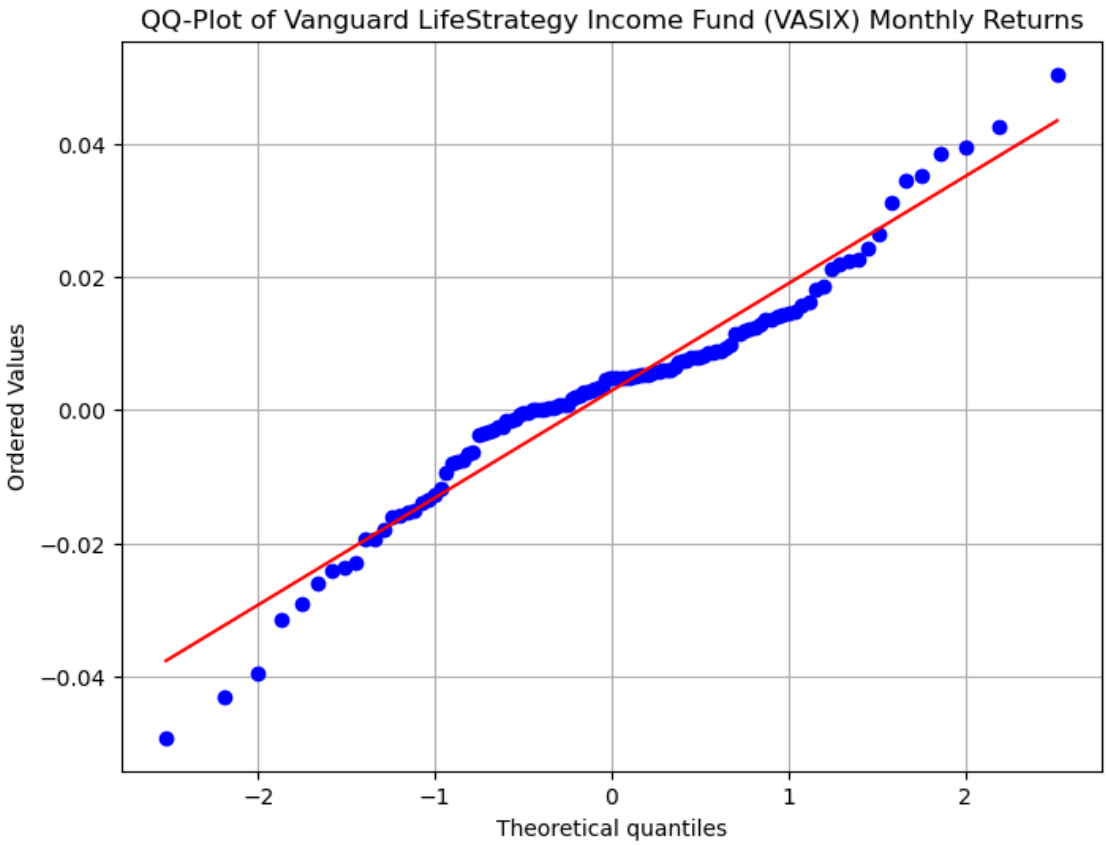

ProShares VIX Mid-Term Futures (VIXM) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
Fitted T-Distribution (Bootstrap 1)
Fitted Cauchy Distribution (Bootstrap 1)
Fitted Normal Distribution (Bootstrap 1)
ProShares VIX Mid-Term Futures (VIXM) Returns
Density
Monthly Returns
0
1
2
3
4
5
6
7
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
0.5
0.6

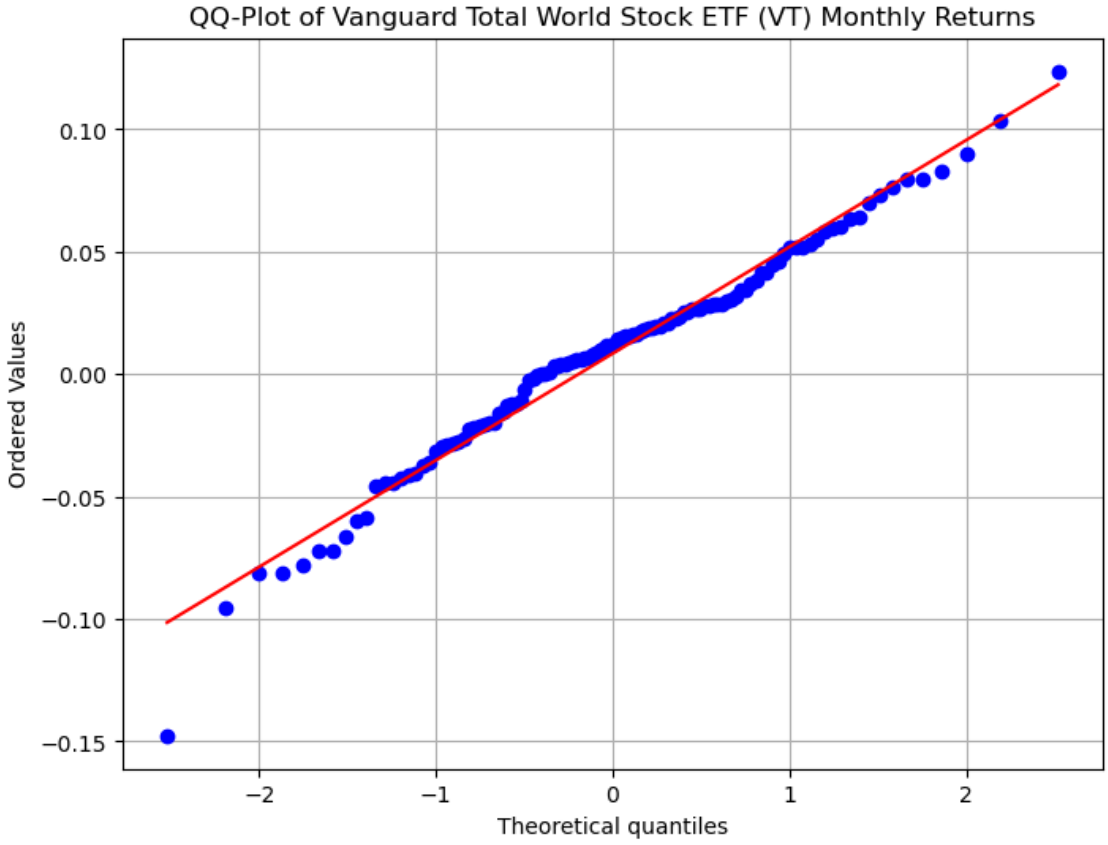

Vanguard LifeStrategy Income Fund (VASIX) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
Fitted T-Distribution (Bootstrap 1)
Fitted Cauchy Distribution (Bootstrap 1)
Fitted Normal Distribution (Bootstrap 1)
Vanguard LifeStrategy Income Fund (VASIX) Returns
Density
Monthly Returns
Vanguard Total World Stock ETF (VT) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
Vanguard Total World Stock ETF (VT) Returns
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ) Returns
AQR Diversified Arbitrage I (ADAIX) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
AQR Diversified Arbitrage I (ADAIX) Returns
iShares Gold Trust (IAU) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
iShares Gold Trust (IAU) Returns
Bitcoin Market Price USD (^BTC) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
Bitcoin Market Price USD (^BTC) Returns
AQR Risk-Balanced Commodities Strategy I (ARCIX) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
AQR Risk-Balanced Commodities Strategy I (ARCIX) Returns
AQR Long-Short Equity I (QLEIX) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
AQR Long-Short Equity I (QLEIX) Returns
AQR Style Premia Alternative I (QSPIX) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
AQR Style Premia Alternative I (QSPIX) Returns
AQR Equity Market Neutral I (QMNIX) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
AQR Equity Market Neutral I (QMNIX) Returns
AQR Macro Opportunities I (QGMIX) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
AQR Macro Opportunities I (QGMIX) Returns
AGF U.S. Market Neutral Anti-Beta (BTAL) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
AGF U.S. Market Neutral Anti-Beta (BTAL) Returns
AQR Managed Futures Strategy HV I (QMHIX) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
AQR Managed Futures Strategy HV I (QMHIX) Returns
Invesco DB US Dollar Bullish (UUP) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
Invesco DB US Dollar Bullish (UUP) Returns
ProShares VIX Mid-Term Futures (VIXM) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
ProShares VIX Mid-Term Futures (VIXM) Returns

17 QQ Plots of Asset Returns (Figure 9)

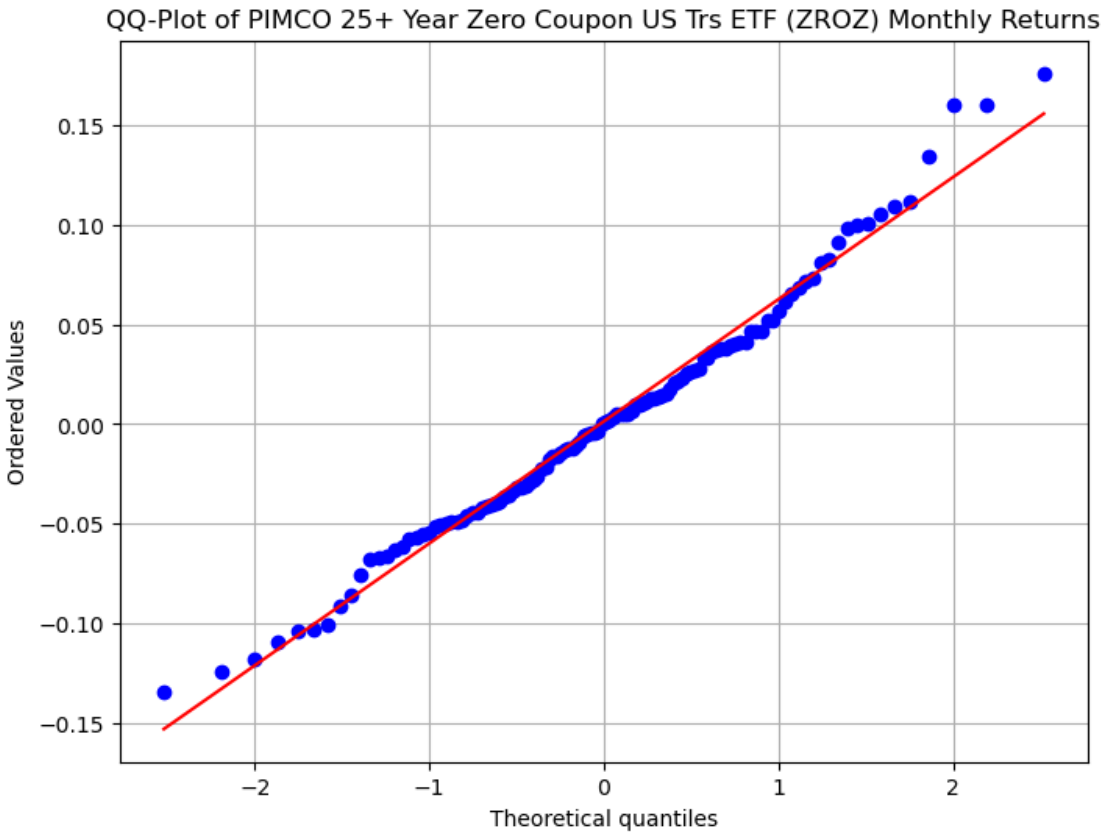
[24]:
```
import os
import matplotlib.pyplot as plt
import scipy.stats as stats

# Comment-based snippet name for the folder
snippet_name = "QQ_PLOTS"

# Create the directory to save plots
output_folder = snippet_name
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# Create a list to hold individual QQ plot data
figures_data = []

# Iterate over all assets excluding the 'Date' column
for asset in data.columns:
    if asset != 'Date':  # Skip the 'Date' column
        # Generate QQ plot data
        (osm, osr), (slope, intercept, r) = stats.probplot(data[asset],␣
 ↪dist="norm")
        figures_data.append((osm, osr, slope, intercept, r, asset))

        # Plot the QQ-Plot
        fig, ax = plt.subplots(figsize=(8, 6))
        stats.probplot(data[asset], dist="norm", plot=ax)
        ax.set_title(f"QQ-Plot of {asset} Monthly Returns")
        ax.grid(True)

        # Save the figure as a PNG file inside the created folder
        fig.savefig(os.path.join(output_folder, f"{asset}_qq_plot.png"),␣
 ↪bbox_inches='tight', dpi=300)  # Save with high resolution

        # Show the plot
        plt.show()

        # Close the figure to save memory
        plt.close(fig)

# Create a new figure to combine all QQ plots into a single image
combined_fig, combined_axes = plt.subplots(len(figures_data) // 2 +␣
 ↪len(figures_data) % 2, 2, figsize=(16, 22))  # Increase figure size for␣
 ↪better readability
combined_axes = combined_axes.flatten()
```

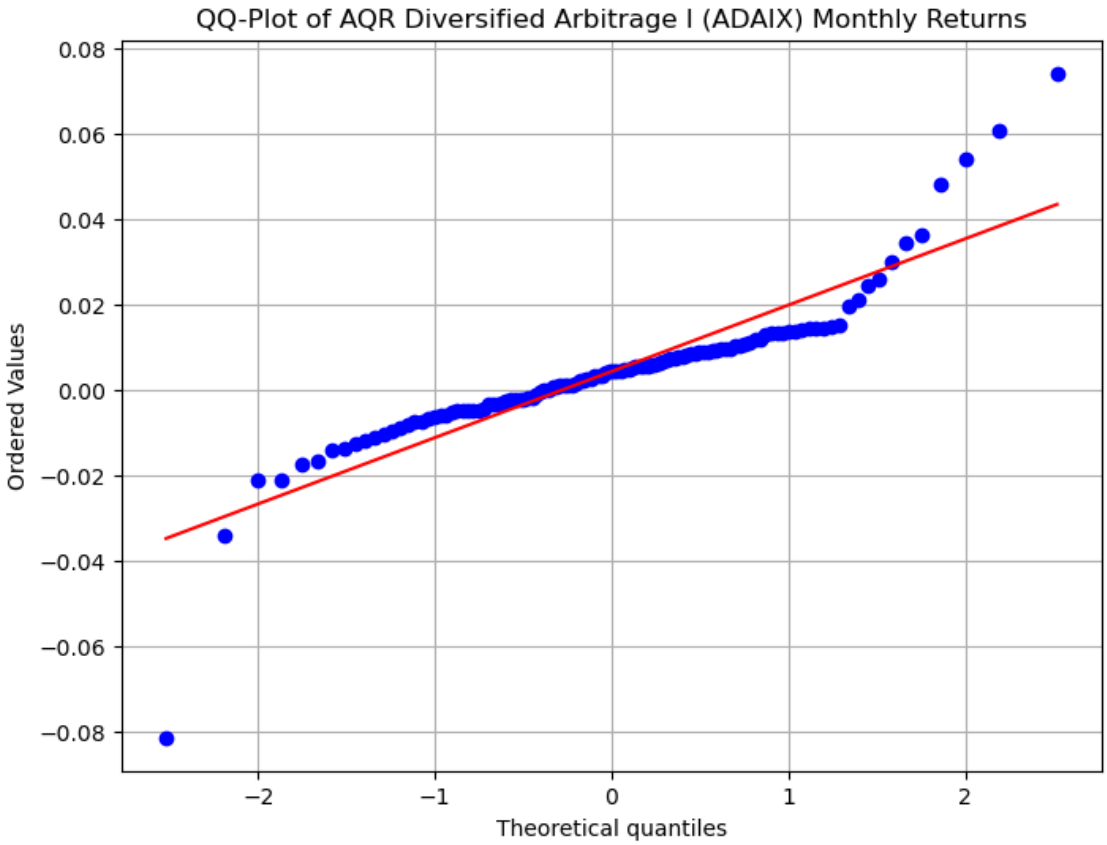
```
# Add each individual QQ plot to the combined figure without distortion
for i, (osm, osr, slope, intercept, r, asset) in enumerate(figures_data):
    ax = combined_axes[i]
    ax.plot(osm, osr, 'o')
    ax.plot(osm, slope * osm + intercept, 'r-', lw=2)
    ax.set_title(f"QQ-Plot of {asset} Monthly Returns", fontsize=10)
    ax.set_xlabel("Theoretical Quantiles", fontsize=8)
    ax.set_ylabel("Ordered Values", fontsize=8)
    ax.tick_params(axis='x', rotation=45, labelsize=8)
    ax.tick_params(axis='y', labelsize=8)
    ax.grid(True)

# Hide any unused subplots
for j in range(len(figures_data), len(combined_axes)):
    combined_fig.delaxes(combined_axes[j])

# Adjust layout to prevent overlap
combined_fig.tight_layout()

# Save the combined figure as a PNG file
combined_fig.savefig(os.path.join(output_folder, "combined_qq_plots.png"),␣
 ↪bbox_inches='tight', dpi=300)  # Save with high resolution

# Display the combined plot in Jupyter Notebook
plt.show()
```



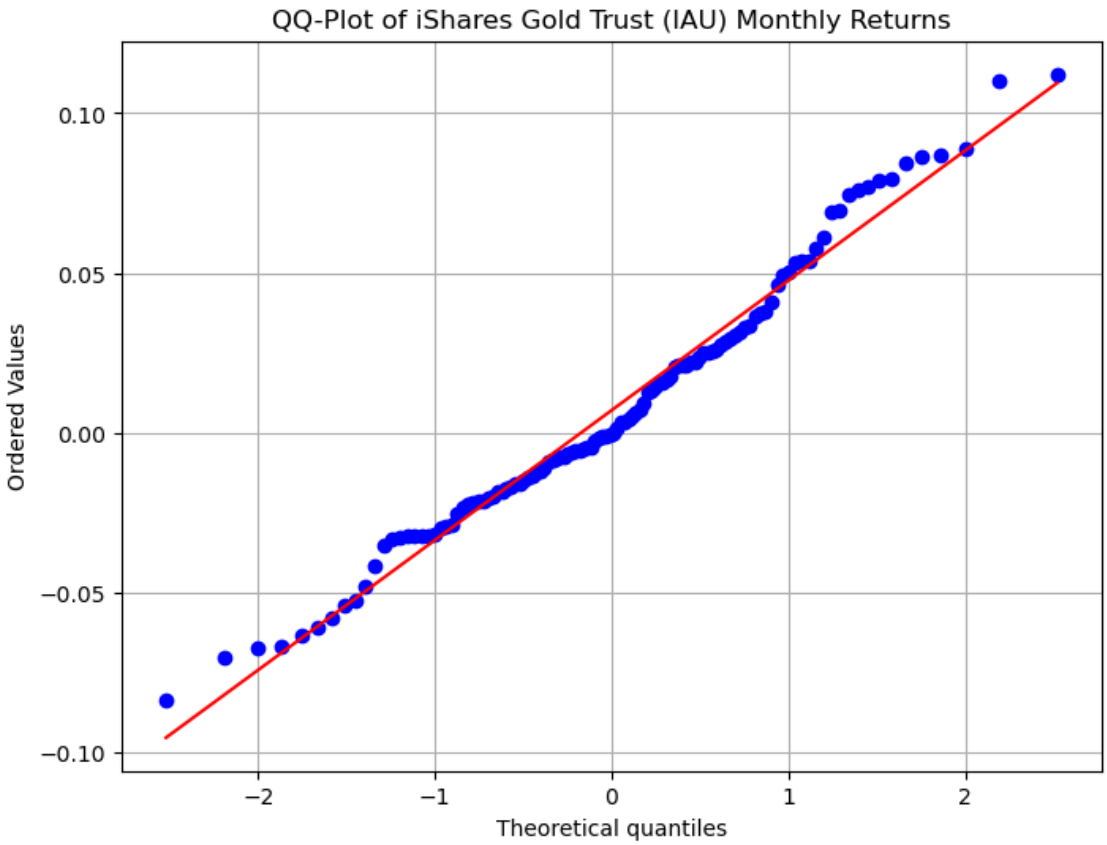


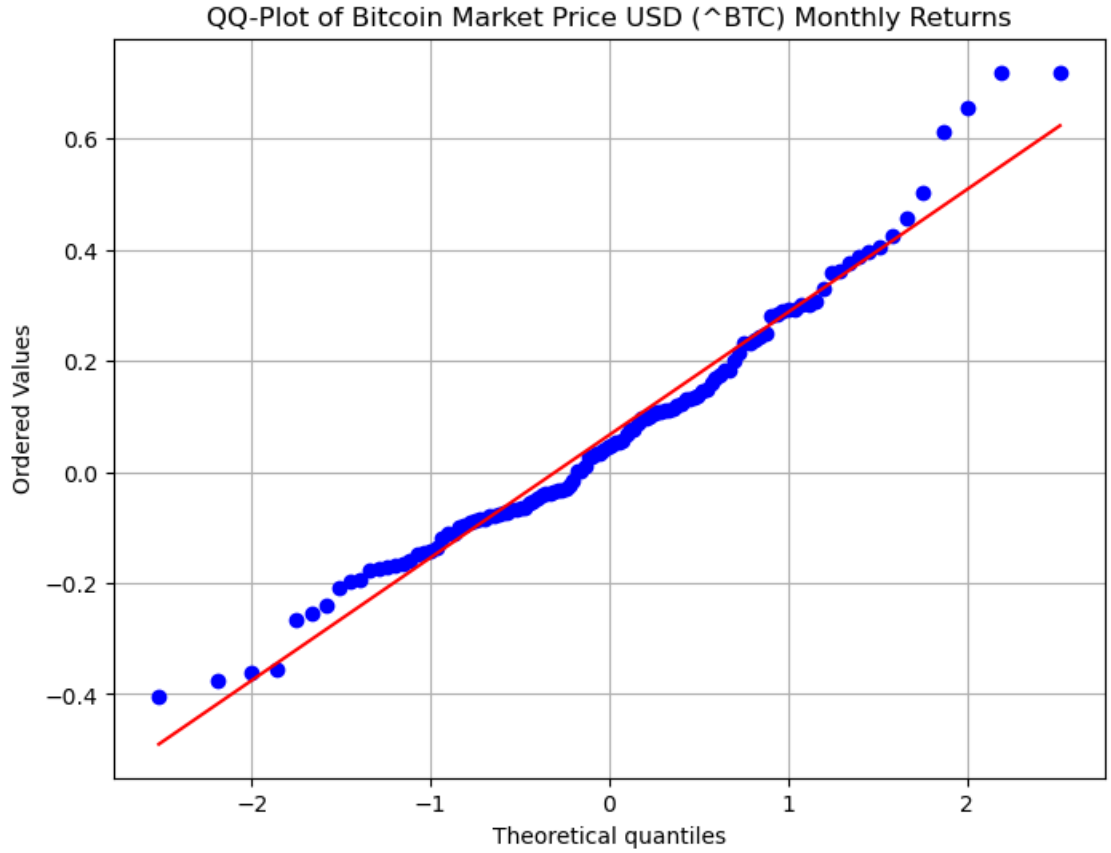
Appendix B1 - Ian Exploratory Data Analysis



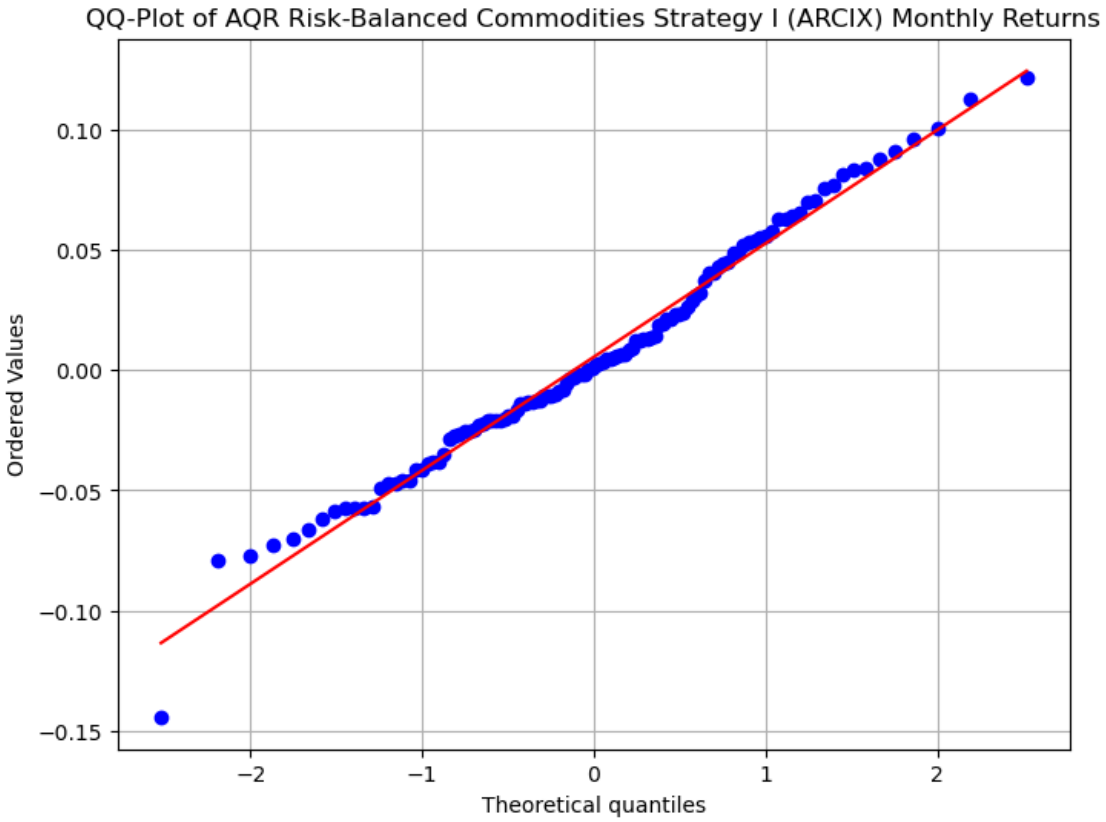


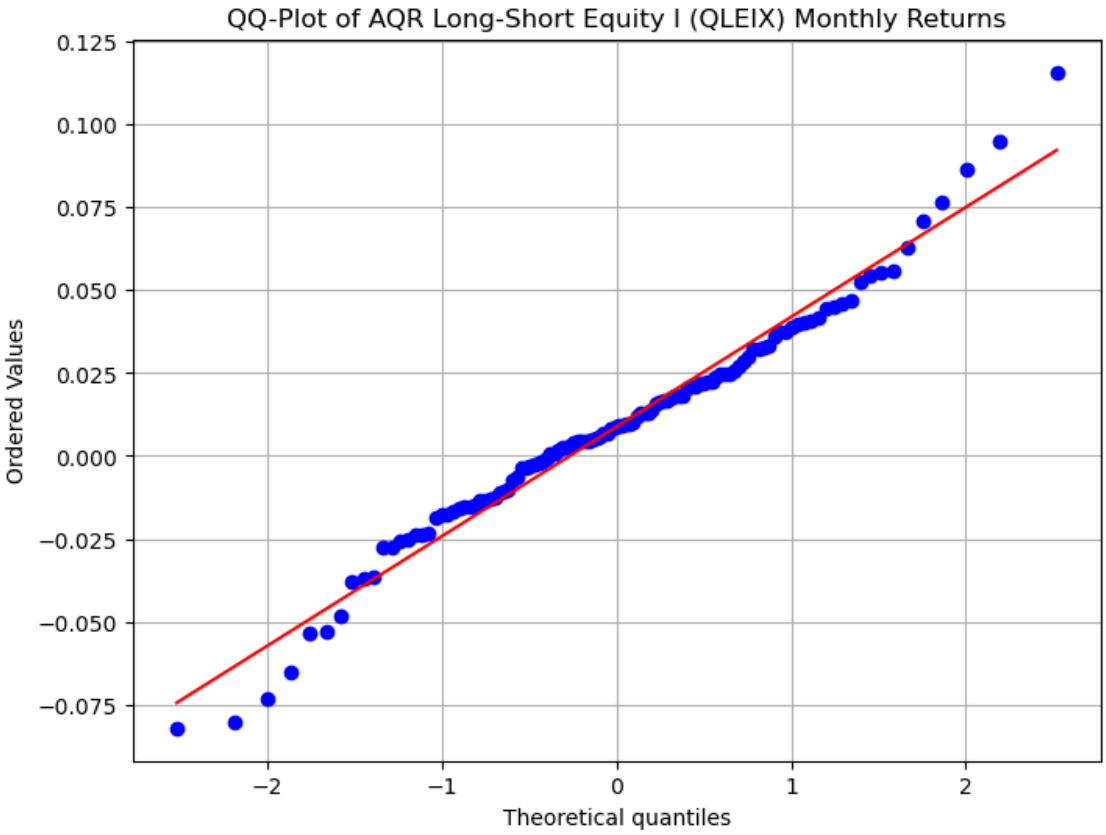
Appendix B1 - Ian Exploratory Data Analysis



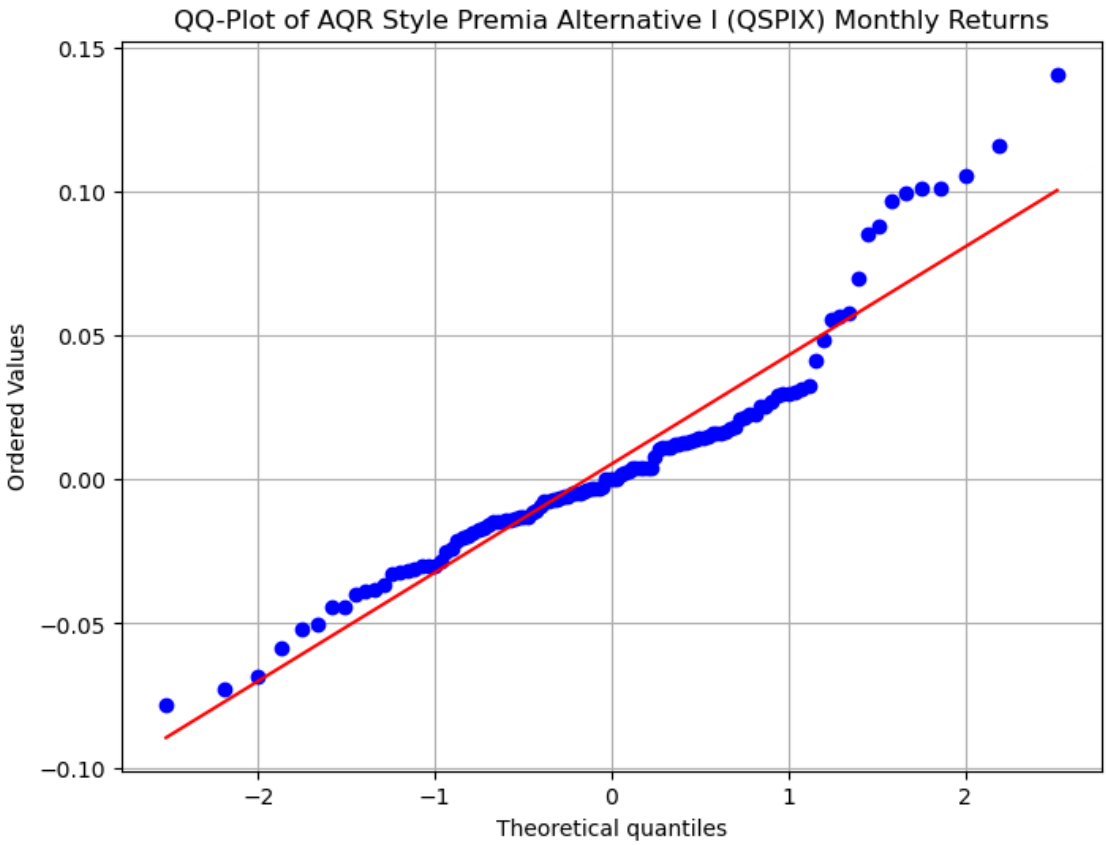


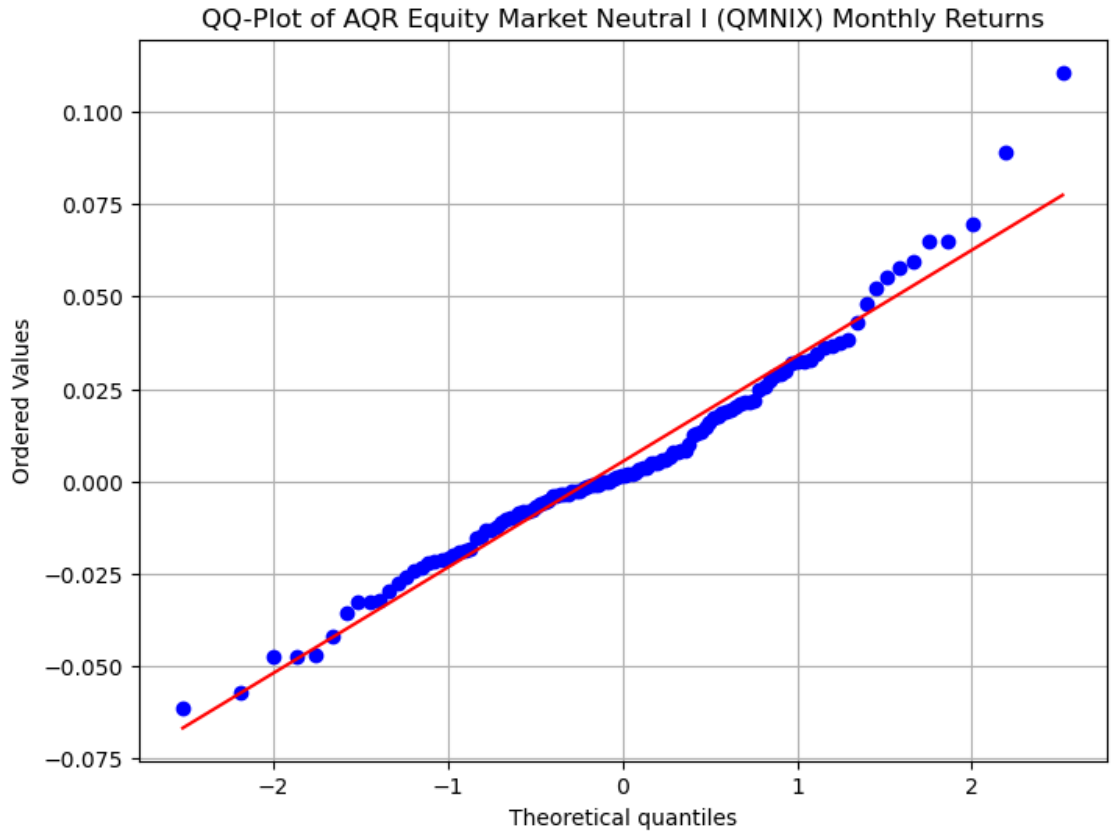
Appendix B1 - Ian Exploratory Data Analysis



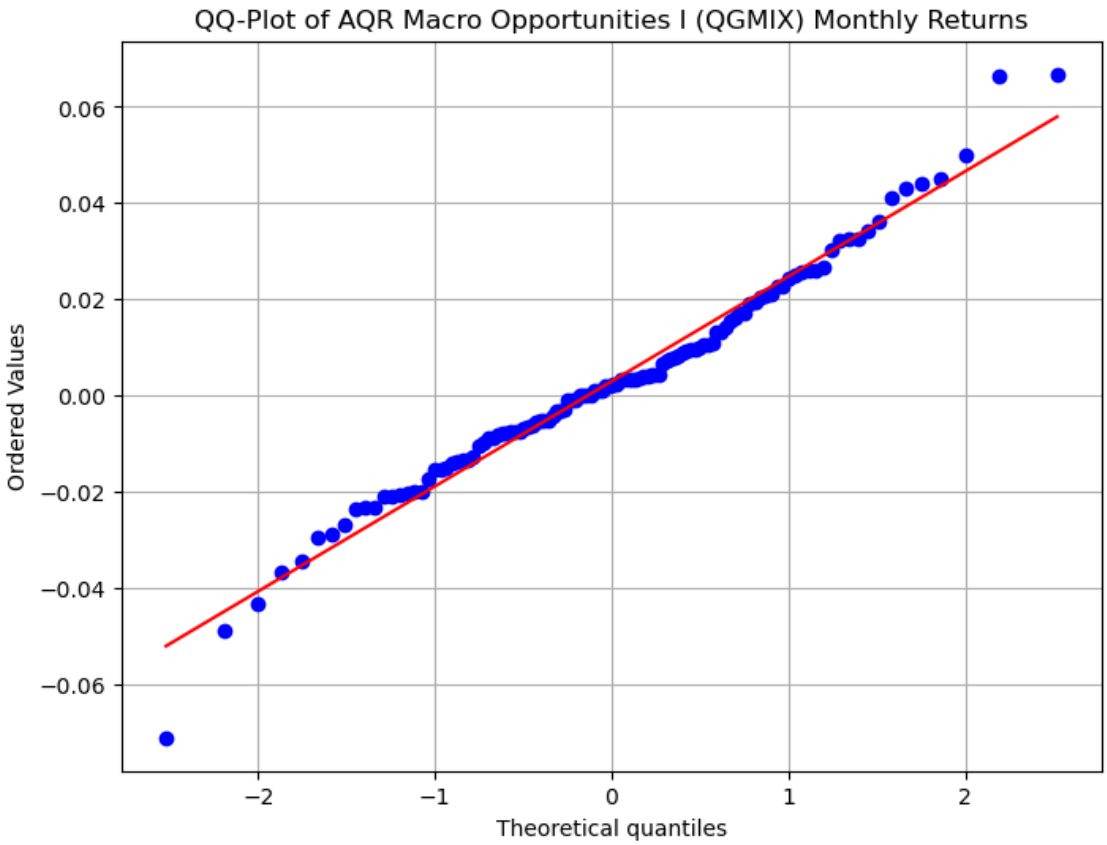


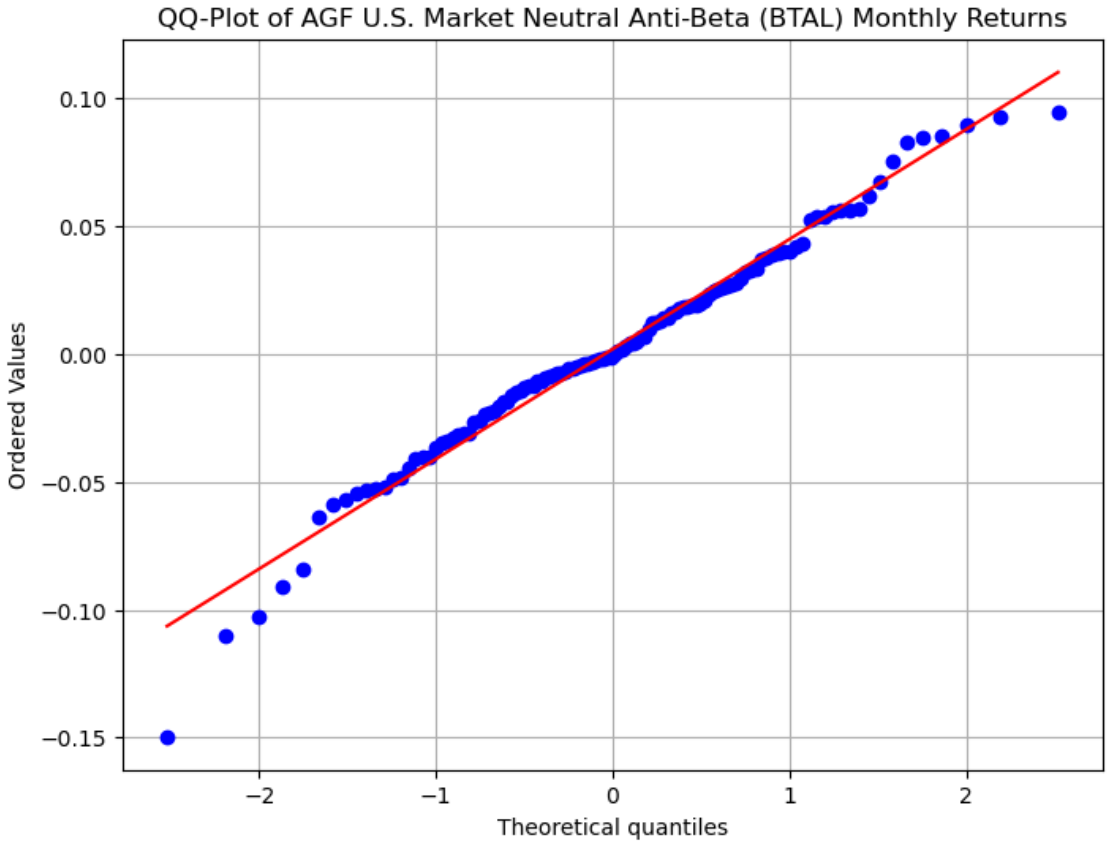
Appendix B1 - Ian Exploratory Data Analysis



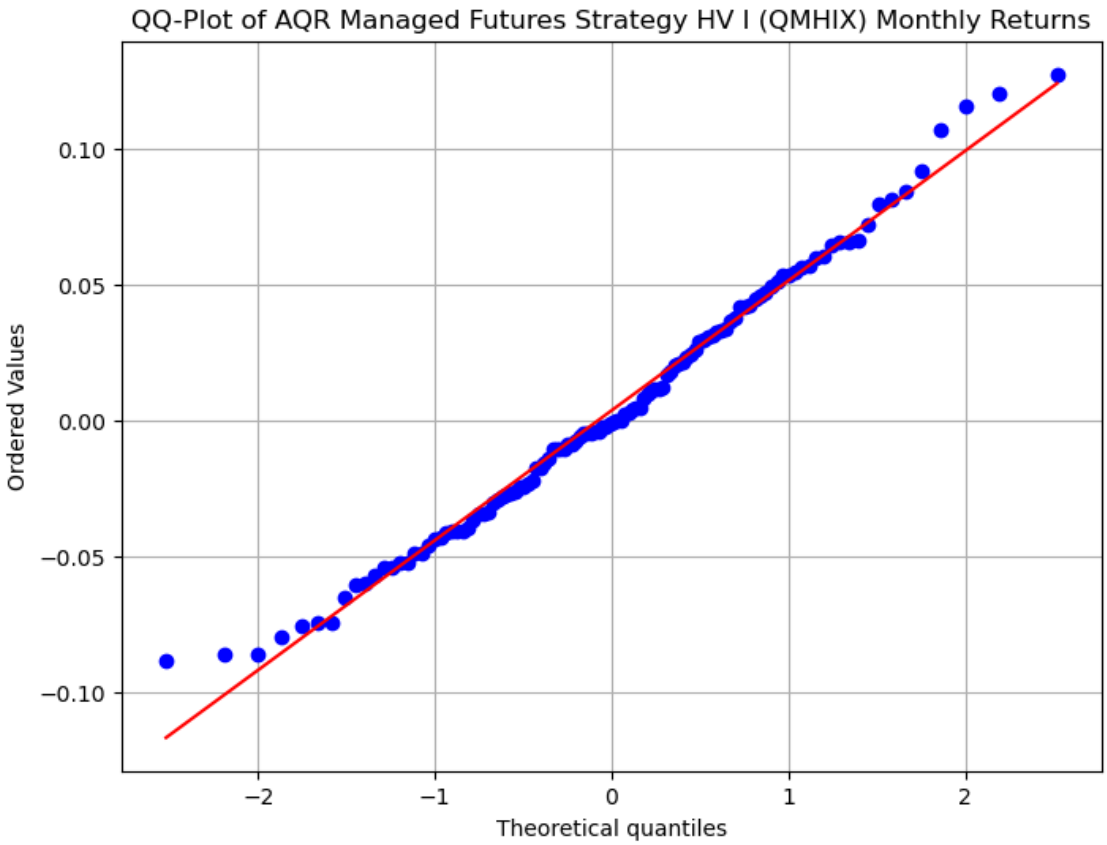


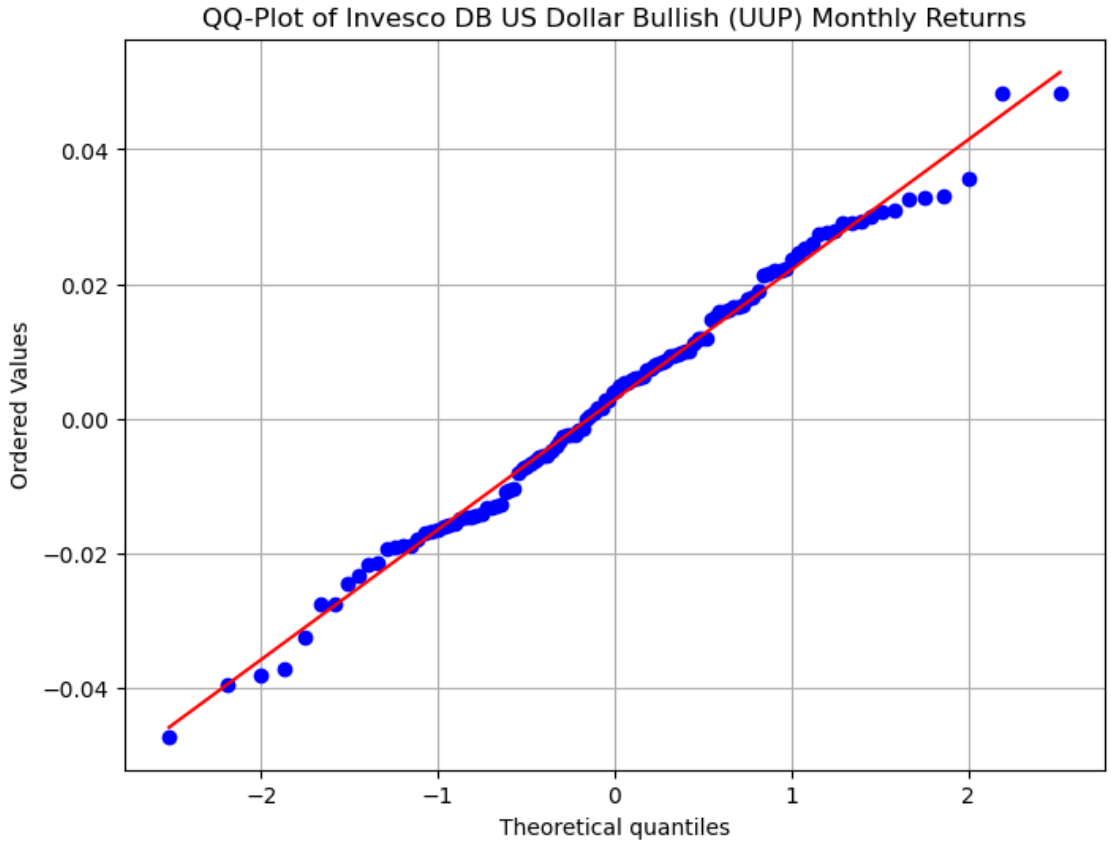
Appendix B1 - Ian Exploratory Data Analysis



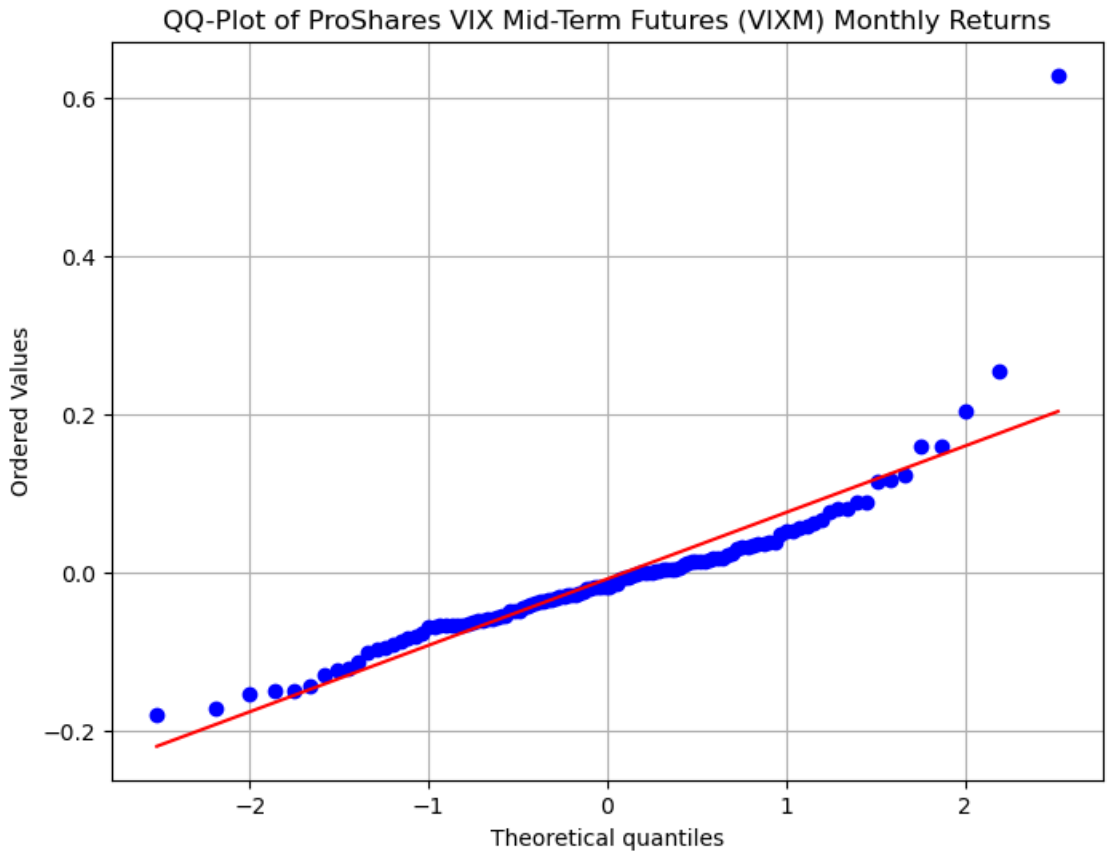


Appendix B1 - Ian Exploratory Data Analysis

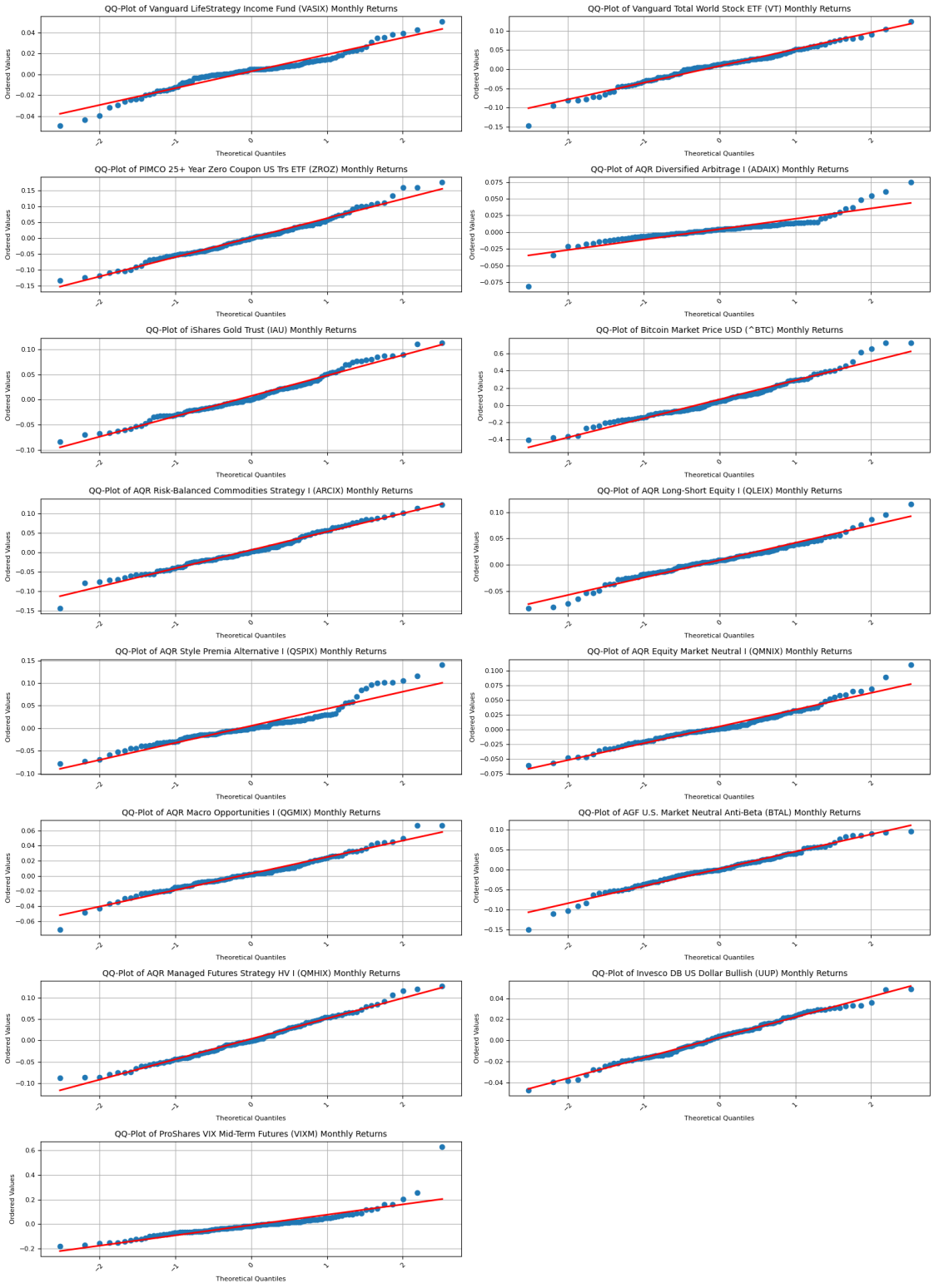




Appendix B1 - Ian Exploratory Data Analysis



Appendix B1 - Ian Exploratory Data Analysis



Appendix B1 - Ian Exploratory Data Analysis

18 Quantitative Goodness of Fit Tests (AIC, BIC, K-S) (Table 7)

[225]:
```
import os
import numpy as np
import pandas as pd
import dataframe_image as dfi  # For exporting DataFrames as images
from scipy.stats import t, cauchy, norm, kstest

# Comment-based snippet name for the folder
snippet_name = "AIC_BIC_KS_TABLE"

# Create the directory to save the table
output_folder = os.path.join(os.getcwd(), snippet_name)
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# Function to compute AIC and BIC
def calculate_aic_bic(log_likelihood, num_params, num_data_points):
    aic = 2 * num_params - 2 * log_likelihood
    bic = np.log(num_data_points) * num_params - 2 * log_likelihood
    return aic, bic

# Initialize results list
results_list = []

# Iterate over each asset (excluding 'Date')
for asset in data.columns:
    if asset != 'Date':  # Skip 'Date'
        returns = data[asset]
        n = len(returns)

        # Fit T-Distribution
        df_t, loc_t, scale_t = t.fit(returns)
        ll_t = np.sum(np.log(t.pdf(returns, df_t, loc_t, scale_t)))
        aic_t, bic_t = calculate_aic_bic(ll_t, 3, n)
        ks_t = kstest(returns, 't', args=(df_t, loc_t, scale_t))

        # Fit Cauchy Distribution
        loc_cauchy, scale_cauchy = cauchy.fit(returns)
        ll_cauchy = np.sum(np.log(cauchy.pdf(returns, loc_cauchy,␣
 ↪scale_cauchy)))
        aic_cauchy, bic_cauchy = calculate_aic_bic(ll_cauchy, 2, n)
        ks_cauchy = kstest(returns, 'cauchy', args=(loc_cauchy, scale_cauchy))

        # Fit Normal Distribution
        mu_normal, std_normal = norm.fit(returns)
        ll_normal = np.sum(np.log(norm.pdf(returns, mu_normal, std_normal)))
```

```
        aic_normal, bic_normal = calculate_aic_bic(ll_normal, 2, n)
        ks_normal = kstest(returns, 'norm', args=(mu_normal, std_normal))

        # Collect results for this asset
        results_list.append({
            'Asset': asset,
            'AIC_T': aic_t,
            'AIC_Cauchy': aic_cauchy,
            'AIC_Normal': aic_normal,
            'BIC_T': bic_t,
            'BIC_Cauchy': bic_cauchy,
            'BIC_Normal': bic_normal,
            'K-S_T': ks_t.statistic,
            'K-S_Cauchy': ks_cauchy.statistic,
            'K-S_Normal': ks_normal.statistic
        })

# Convert results to DataFrame
results_df = pd.DataFrame(results_list)

# Ensure the AIC columns are in the correct format
results_df[['AIC_T', 'AIC_Cauchy', 'AIC_Normal']] = results_df[['AIC_T',␣
 ↪'AIC_Cauchy', 'AIC_Normal']].astype(float)

# Limit the display to 3 decimal places
pd.options.display.float_format = '{:.3f}'.format

# Function to highlight the best fit based on AIC, BIC, and K-S separately
def highlight_best_fit(row):
    aic_cols = ['AIC_T', 'AIC_Cauchy', 'AIC_Normal']
    bic_cols = ['BIC_T', 'BIC_Cauchy', 'BIC_Normal']
    ks_cols = ['K-S_T', 'K-S_Cauchy', 'K-S_Normal']

    min_aic = row[aic_cols].min()
    min_bic = row[bic_cols].min()
    min_ks = row[ks_cols].min()

    def highlight(val, col_type):
        if col_type == 'AIC' and val == min_aic:
            return 'background-color: lightgreen; font-weight: bold'
        elif col_type == 'BIC' and val == min_bic:
            return 'background-color: lightblue; font-weight: bold'
        elif col_type == 'K-S' and val == min_ks:
            return 'background-color: lightyellow; font-weight: bold'
        else:
            return ''
```

```
    return [
        highlight(val, 'AIC') if col in aic_cols else
        highlight(val, 'BIC') if col in bic_cols else
        highlight(val, 'K-S') if col in ks_cols else ''
        for col, val in row.items()
    ]

# Apply formatting to highlight the best fit for each metric group
formatted_results = results_df.style.apply(highlight_best_fit, axis=1).
 ↪format(precision=3)

# Save the formatted table as a PNG file inside the created folder
dfi.export(formatted_results, os.path.join(output_folder, "aic_bic_ks_table.
 ↪png"))

# Optionally, display the formatted table in Jupyter
formatted_results
```

[225]: <pandas.io.formats.style.Styler at 0x3131d1940>

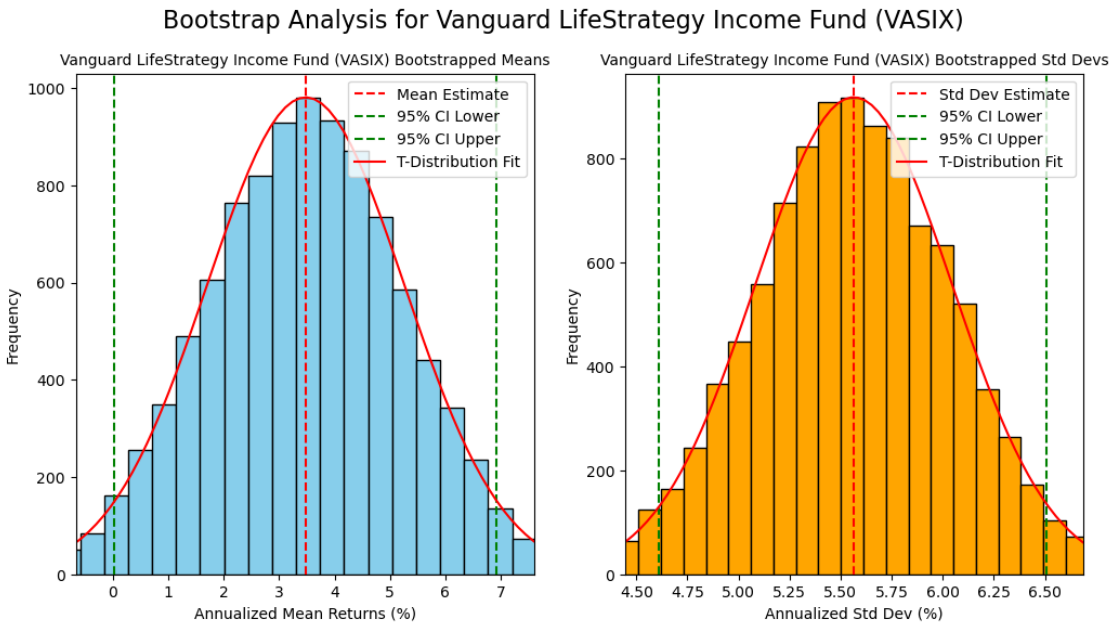
19 Plots of Non-Parametric Bootstraps of Means and Standard Deviations (Not an Exhibit in the Report)

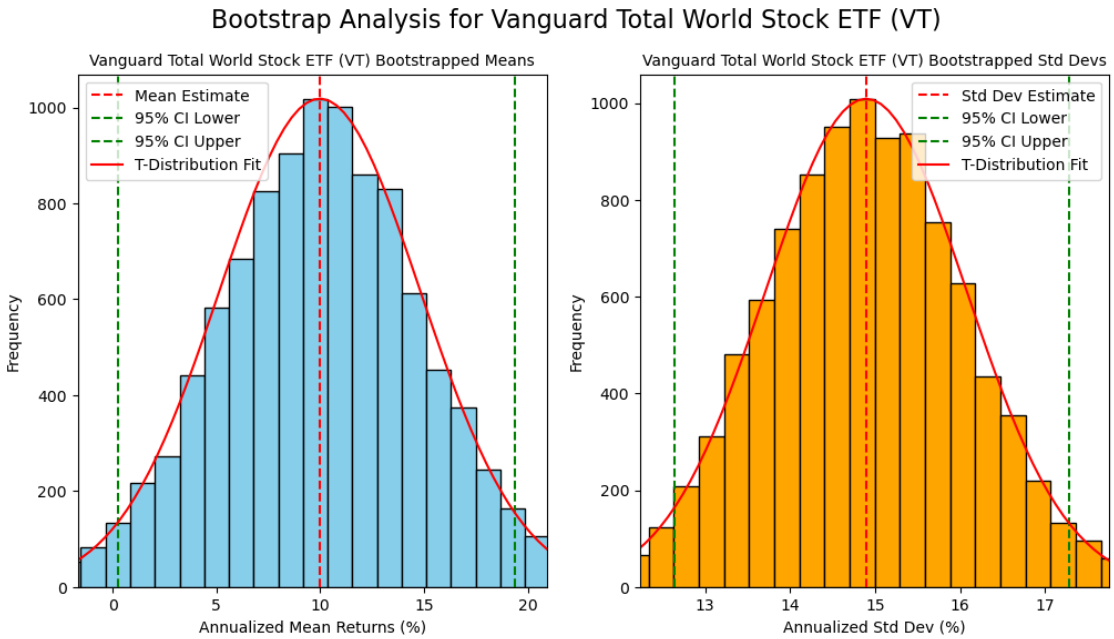
[30]:
```
import os
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import t
from PIL import Image

# Comment-based snippet name for the folder
snippet_name = "NON_PARAMETRIC_BOOTSTRAP_PLOTS"

# Create the directory to save the plots
output_folder = snippet_name
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# Check if 'Date' column exists and drop it if present
if 'Date' in data.columns:
    asset_columns = data.columns.drop('Date')
else:
    asset_columns = data.columns

# Number of bootstrap iterations and seed for reproducibility
n_iterations = 10000
```

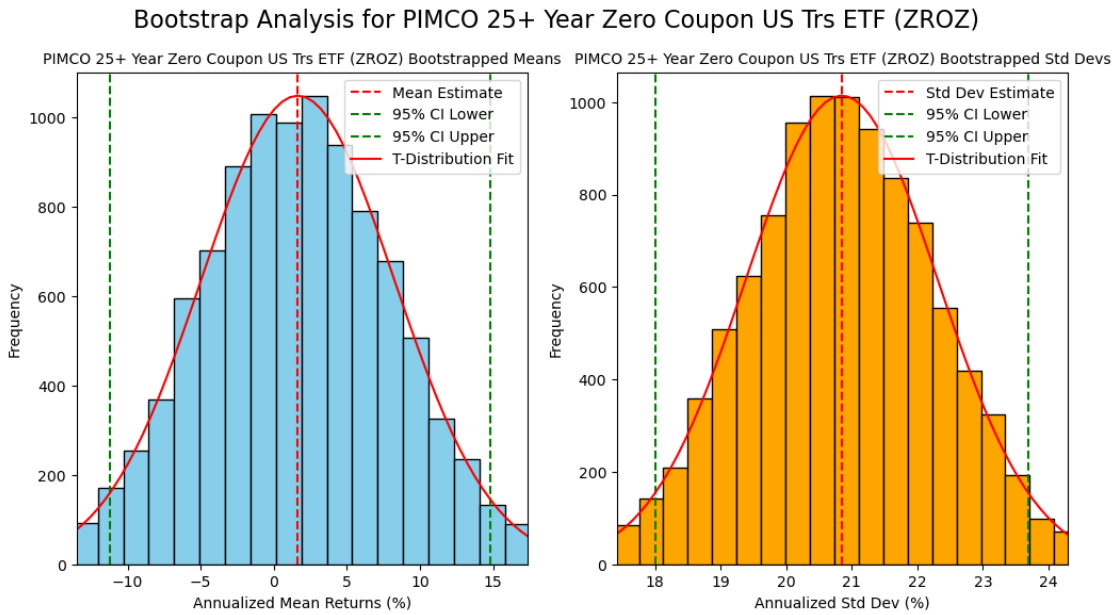
```
np.random.seed(42)  # Set seed for reproducibility

# Determine the number of periods per year (e.g., monthly data)
periods_per_year = 12  # Adjust according to your data frequency

# Create a list to hold individual figures data
figures_paths = []

# Loop through each asset in the dataset (excluding the 'Date' column)
for asset in asset_columns:
    asset_returns = data[asset].dropna()
    n_size = len(asset_returns)

    # Arrays to hold bootstrap estimates
    bootstrap_means = np.zeros(n_iterations)
    bootstrap_vars = np.zeros(n_iterations)

    # Perform bootstrap resampling
    for i in range(n_iterations):
        bootstrap_sample = np.random.choice(asset_returns, size=n_size,␣
↪replace=True)
        bootstrap_means[i] = np.mean(bootstrap_sample)
        bootstrap_vars[i] = np.var(bootstrap_sample)

    # Annualize the bootstrap means and standard deviations
    annualized_bootstrap_means = bootstrap_means * periods_per_year * 100  #␣
↪Annualize means and convert to percentage
    annualized_bootstrap_std_devs = np.sqrt(bootstrap_vars) * np.
↪sqrt(periods_per_year) * 100  # Annualize std dev and convert to percentage

    # Fit a t-distribution to the annualized means and std devs
    df_means, loc_means, scale_means = t.fit(annualized_bootstrap_means)
    df_stddev, loc_stddev, scale_stddev = t.fit(annualized_bootstrap_std_devs)

    # Calculate 95% confidence intervals for the annualized mean returns and␣
↪standard deviations
    mean_ci_lower, mean_ci_upper = np.percentile(annualized_bootstrap_means, [2.
↪5, 97.5])
    std_ci_lower, std_ci_upper = np.percentile(annualized_bootstrap_std_devs,␣
↪[2.5, 97.5])

    # Create a figure for each asset's plots
    fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))
    fig.suptitle(f"Bootstrap Analysis for {asset}", fontsize=16)

    # Plot bootstrapped means (left subplot)
```

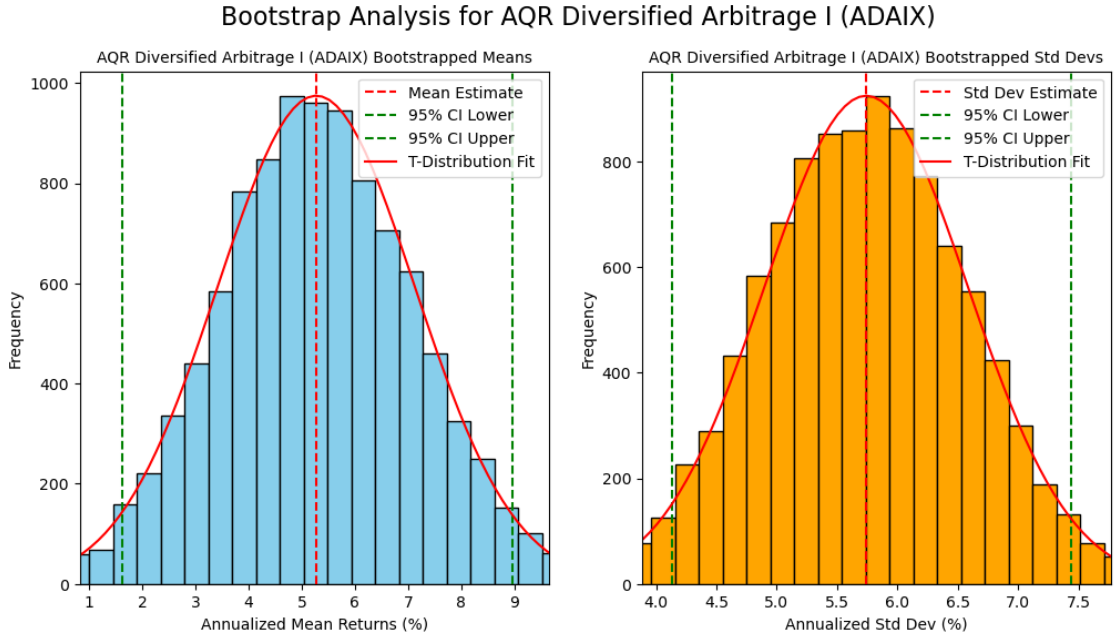
```
    ax_means = axes[0]
    count, bins, _ = ax_means.hist(annualized_bootstrap_means, bins=30,␣
↪color='skyblue', edgecolor='black')  # Get histogram counts
    ax_means.axvline(np.mean(annualized_bootstrap_means), color='red',␣
↪linestyle='--', label="Mean Estimate")
    ax_means.axvline(mean_ci_lower, color='green', linestyle='--', label="95%␣
↪CI Lower")
    ax_means.axvline(mean_ci_upper, color='green', linestyle='--', label="95%␣
↪CI Upper")

    # Overlay t-distribution fit
    x_vals_means = np.linspace(min(annualized_bootstrap_means),␣
↪max(annualized_bootstrap_means), 100)
    pdf_fitted_means = t.pdf(x_vals_means, df_means, loc_means, scale_means)
    scale_factor_means = max(count) / max(pdf_fitted_means)  # Scale the␣
↪t-distribution to match histogram
    ax_means.plot(x_vals_means, pdf_fitted_means * scale_factor_means, 'r-',␣
↪label="T-Distribution Fit")

    ax_means.set_title(f"{asset} Bootstrapped Means", fontsize=10)
    ax_means.set_xlabel("Annualized Mean Returns (%)")
    ax_means.set_ylabel("Frequency")
    ax_means.legend()

    # Dynamically adjust x-axis for the mean returns
    mean_x_min, mean_x_max = np.percentile(annualized_bootstrap_means, [1, 99])
    ax_means.set_xlim(left=mean_x_min, right=mean_x_max)

    # Plot bootstrapped standard deviations (right subplot)
    ax_vars = axes[1]
    count_std, bins_std, _ = ax_vars.hist(annualized_bootstrap_std_devs,␣
↪bins=30, color='orange', edgecolor='black')  # Get histogram counts
    ax_vars.axvline(np.mean(annualized_bootstrap_std_devs), color='red',␣
↪linestyle='--', label="Std Dev Estimate")
    ax_vars.axvline(std_ci_lower, color='green', linestyle='--', label="95% CI␣
↪Lower")
    ax_vars.axvline(std_ci_upper, color='green', linestyle='--', label="95% CI␣
↪Upper")

    # Overlay t-distribution fit
    x_vals_stddev = np.linspace(min(annualized_bootstrap_std_devs),␣
↪max(annualized_bootstrap_std_devs), 100)
    pdf_fitted_stddev = t.pdf(x_vals_stddev, df_stddev, loc_stddev,␣
↪scale_stddev)
    scale_factor_stddev = max(count_std) / max(pdf_fitted_stddev)  # Scale the␣
↪t-distribution to match histogram
```

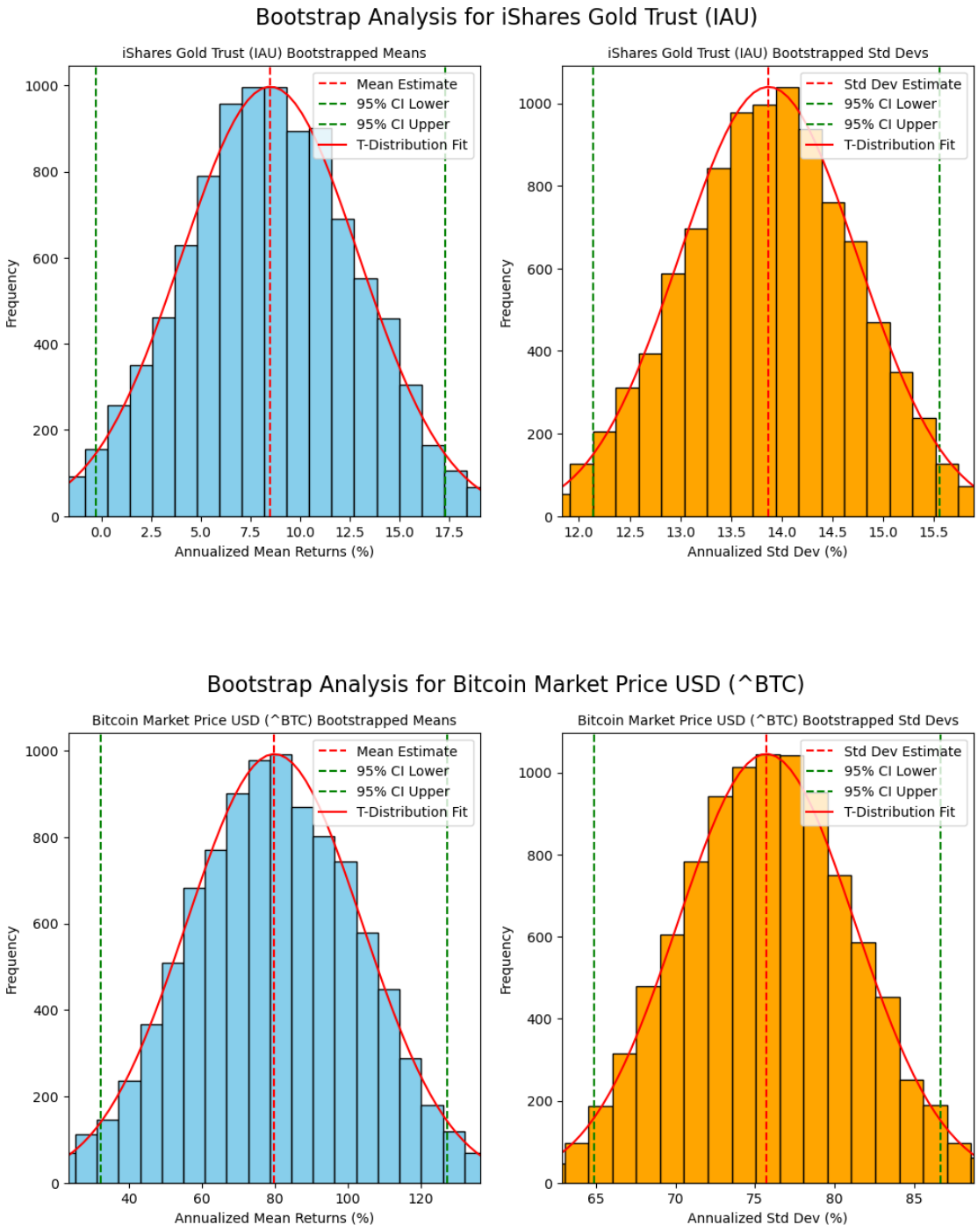
```
    ax_vars.plot(x_vals_stddev, pdf_fitted_stddev * scale_factor_stddev, 'r-',␣
 ↪label="T-Distribution Fit")

    ax_vars.set_title(f"{asset} Bootstrapped Std Devs", fontsize=10)
    ax_vars.set_xlabel("Annualized Std Dev (%)")
    ax_vars.set_ylabel("Frequency")
    ax_vars.legend()

    # Dynamically adjust x-axis for the standard deviations
    std_x_min, std_x_max = np.percentile(annualized_bootstrap_std_devs, [1, 99])
    ax_vars.set_xlim(left=std_x_min, right=std_x_max)

    # Save the figure as a PNG file inside the created folder
    plot_path = os.path.join(output_folder, f"{asset}_bootstrap_plots.png")
    plt.savefig(plot_path, bbox_inches='tight', dpi=300)
    figures_paths.append(plot_path)

    # Display the plots
    plt.show()

# Combine all individual PNGs into a single image with a white background
images = [Image.open(path) for path in figures_paths]
width, height = images[0].size
combined_image = Image.new('RGB', (width * 2, height * ((len(images) + 1) //␣
 ↪2)), (255, 255, 255))

# Paste individual images into the combined image
for idx, image in enumerate(images):
    x_offset = (idx % 2) * width
    y_offset = (idx // 2) * height
    combined_image.paste(image, (x_offset, y_offset))

# Save the combined image
combined_image_path = os.path.join(output_folder, "combined_bootstrap_plots.
 ↪png")
combined_image.save(combined_image_path, format='PNG')

# Display the combined image
combined_image.show()
```

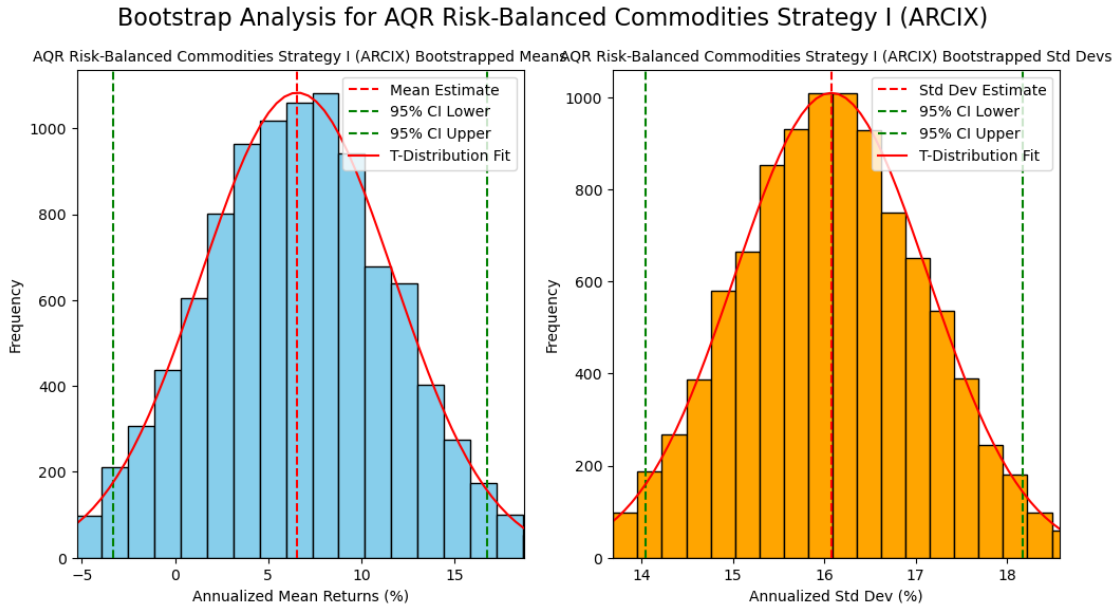

Bootstrap Analysis for Vanguard LifeStrategy Income Fund (VASIX)
Vanguard LifeStrategy Income Fund (VASIX) Bootstrapped Means
Mean Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Mean Returns (%)
Vanguard LifeStrategy Income Fund (VASIX) Bootstrapped Std Devs
Std Dev Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Std Dev (%)

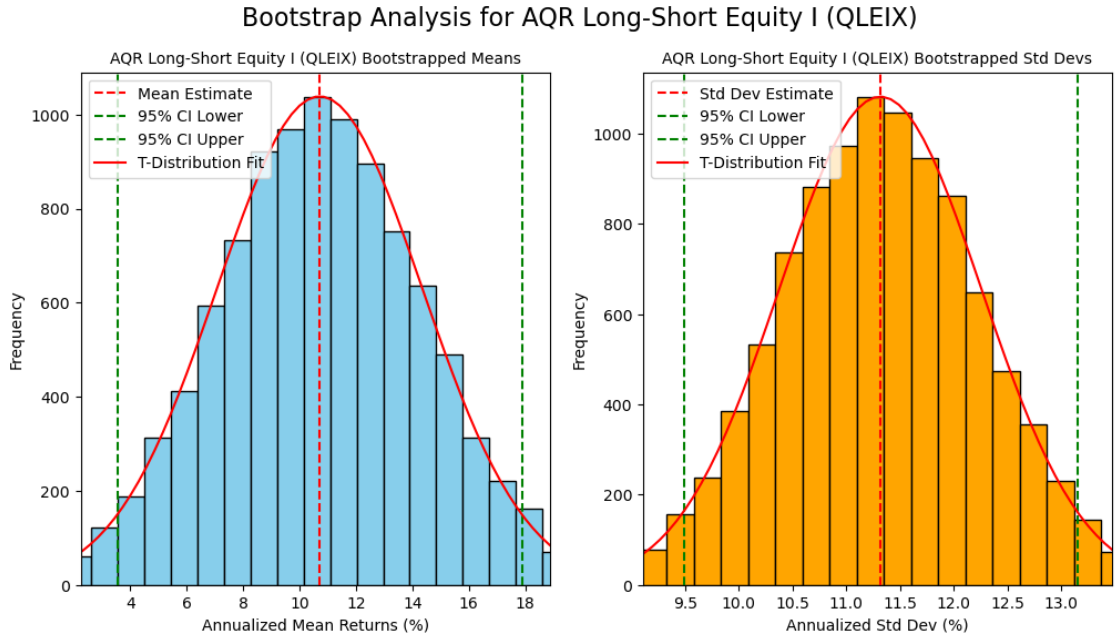

Bootstrap Analysis for Vanguard Total World Stock ETF (VT)
Vanguard Total World Stock ETF (VT) Bootstrapped Means
Mean Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Mean Returns (%)
Vanguard Total World Stock ETF (VT) Bootstrapped Std Devs
Std Dev Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Std Dev (%)

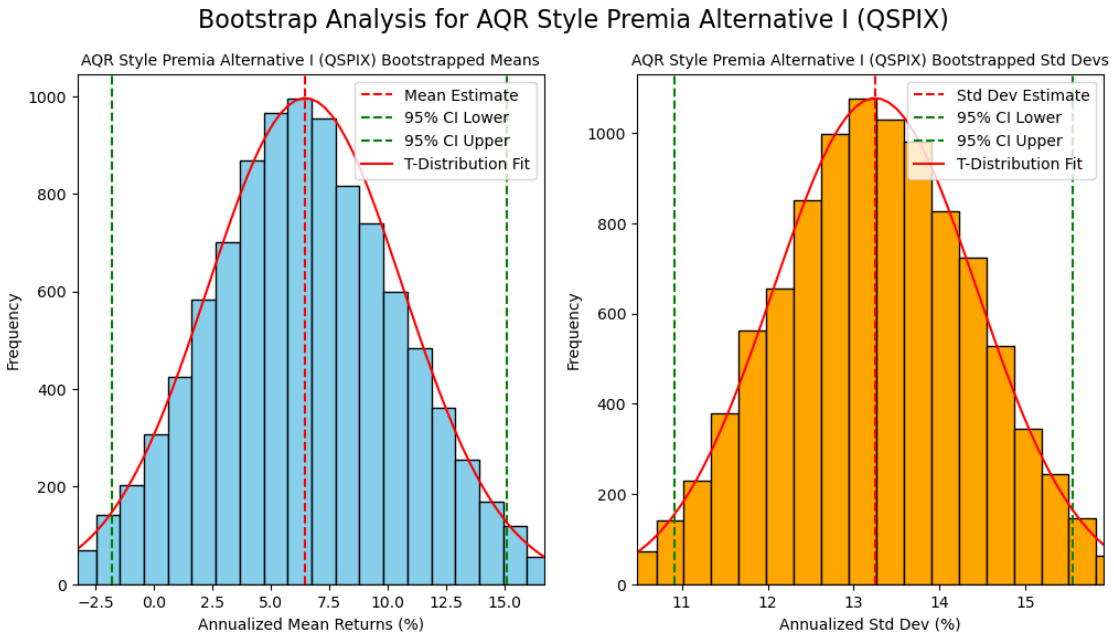

Bootstrap Analysis for PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ) Bootstrapped Means
Mean Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Mean Returns (%)
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ) Bootstrapped Std Devs
Std Dev Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Std Dev (%)

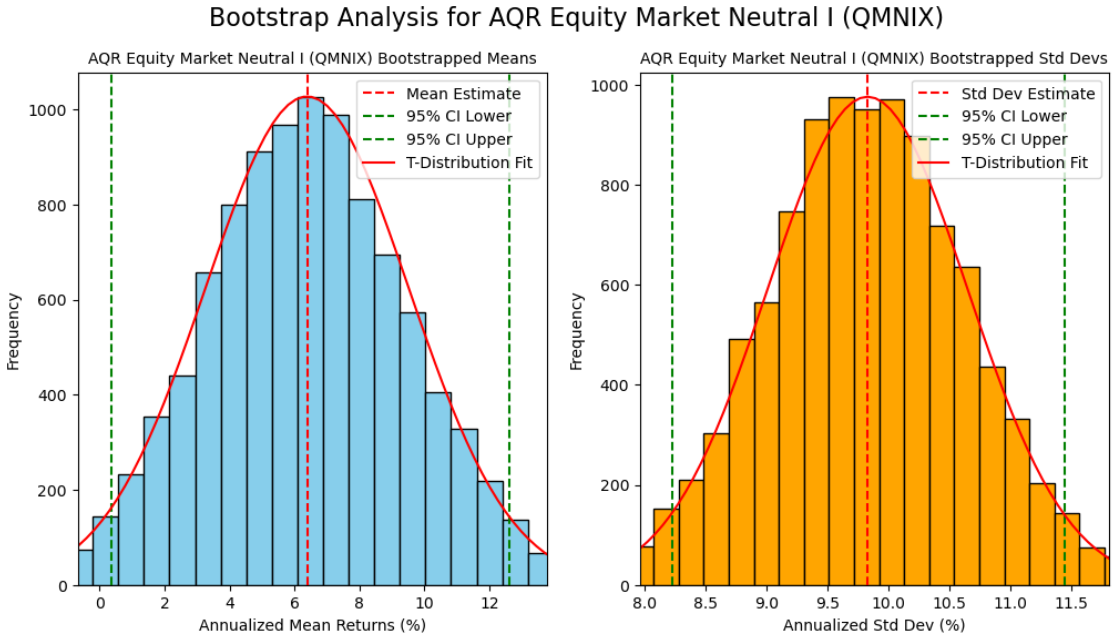

Bootstrap Analysis for AQR Diversified Arbitrage I (ADAIX)
AQR Diversified Arbitrage I (ADAIX) Bootstrapped Means
Mean Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Mean Returns (%)
AQR Diversified Arbitrage I (ADAIX) Bootstrapped Std Devs
Std Dev Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Std Dev (%)


Bootstrap Analysis for iShares Gold Trust (IAU)
iShares Gold Trust (IAU) Bootstrapped Means
Mean Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Mean Returns (%)
iShares Gold Trust (IAU) Bootstrapped Std Devs
Std Dev Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Std Dev (%)
Bootstrap Analysis for Bitcoin Market Price USD (^BTC)
Bitcoin Market Price USD (^BTC) Bootstrapped Means
Mean Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Mean Returns (%)
Bitcoin Market Price USD (^BTC) Bootstrapped Std Devs
Std Dev Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Std Dev (%)

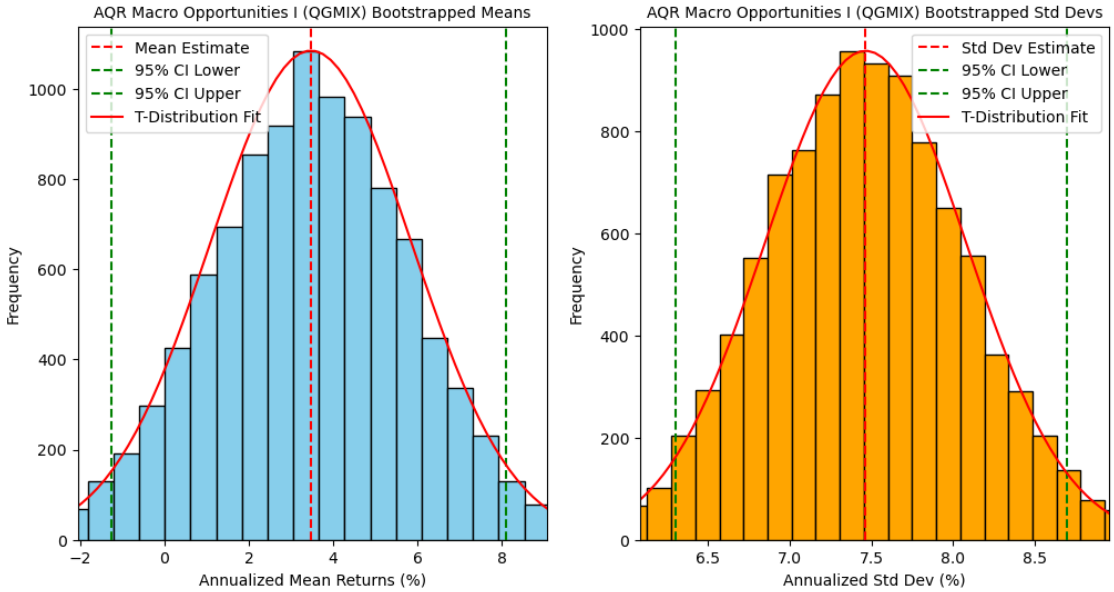

Bootstrap Analysis for AQR Risk-Balanced Commodities Strategy I (ARCIX)
AQR Risk-Balanced Commodities Strategy I (ARCIX) Bootstrapped Means
Mean Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Mean Returns (%)
AQR Risk-Balanced Commodities Strategy I (ARCIX) Bootstrapped Std Devs
Std Dev Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Std Dev (%)


Bootstrap Analysis for AQR Long-Short Equity I (QLEIX)
AQR Long-Short Equity I (QLEIX) Bootstrapped Means
Mean Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Mean Returns (%)
AQR Long-Short Equity I (QLEIX) Bootstrapped Std Devs
Std Dev Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Std Dev (%)

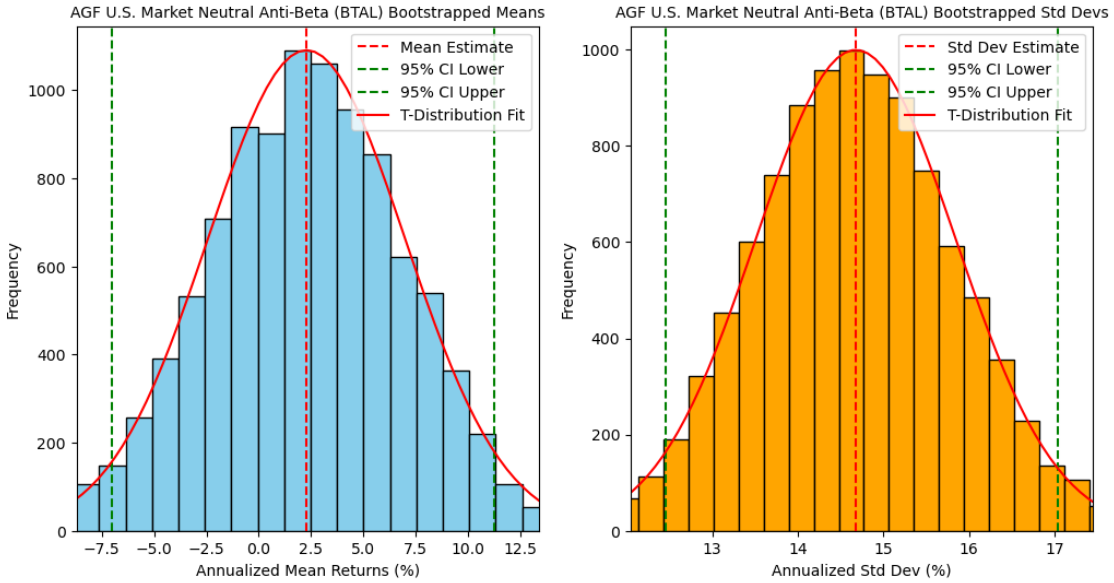

Bootstrap Analysis for AQR Style Premia Alternative I (QSPIX)
AQR Style Premia Alternative I (QSPIX) Bootstrapped Means
Mean Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Mean Returns (%)
AQR Style Premia Alternative I (QSPIX) Bootstrapped Std Devs
Std Dev Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Std Dev (%)

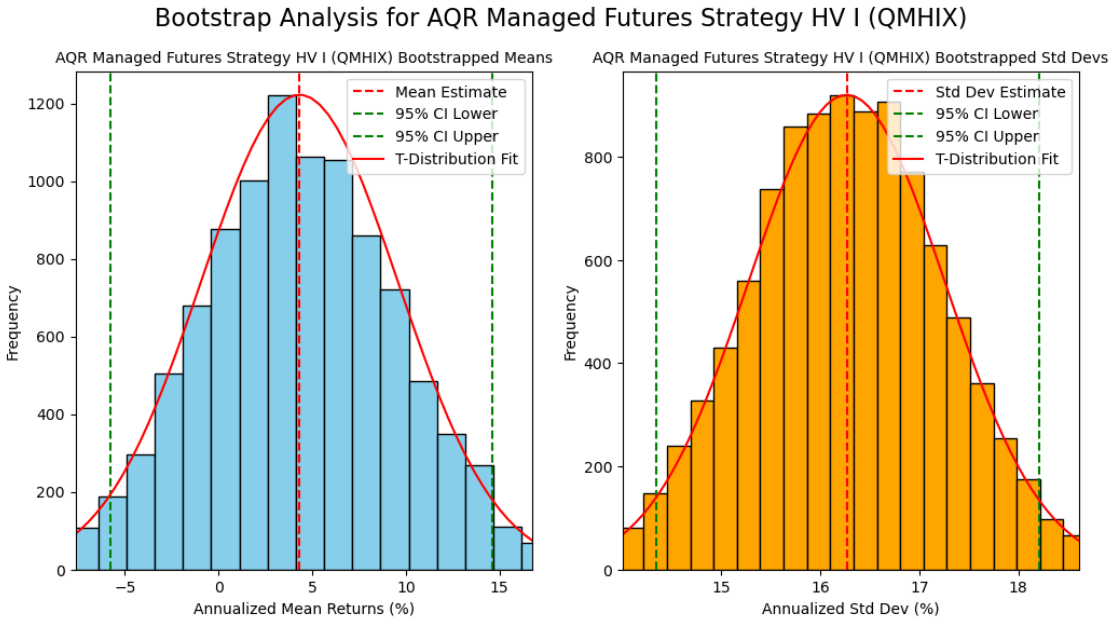

Bootstrap Analysis for AQR Equity Market Neutral I (QMNIX)
AQR Equity Market Neutral I (QMNIX) Bootstrapped Means
Mean Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Mean Returns (%)
AQR Equity Market Neutral I (QMNIX) Bootstrapped Std Devs
Std Dev Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Std Dev (%)

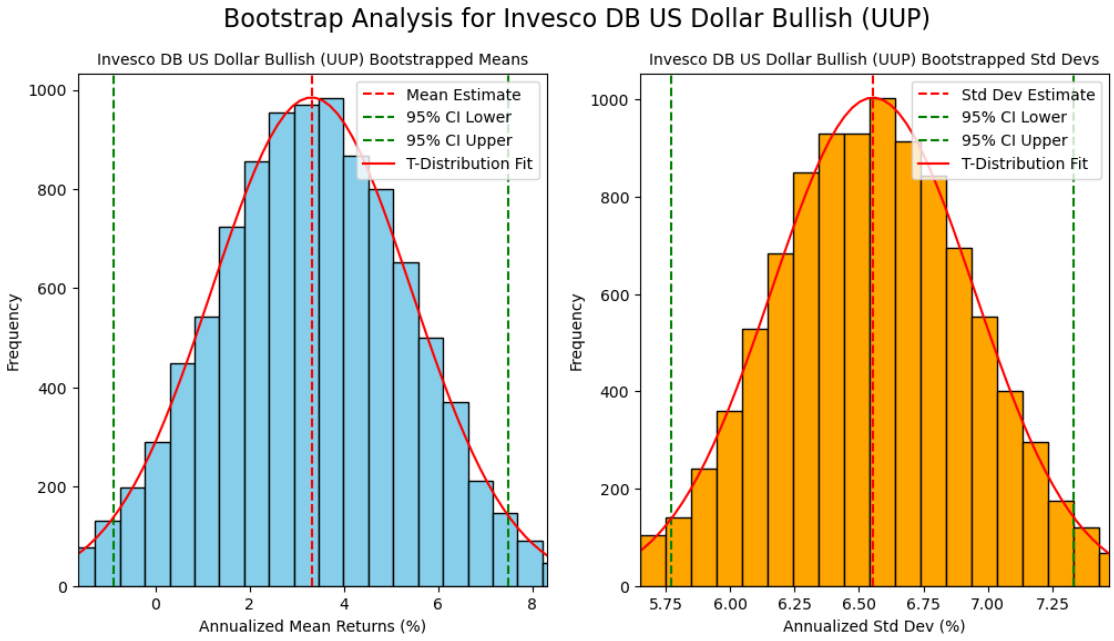
Bootstrap Analysis for AQR Macro Opportunities I (QGMIX)

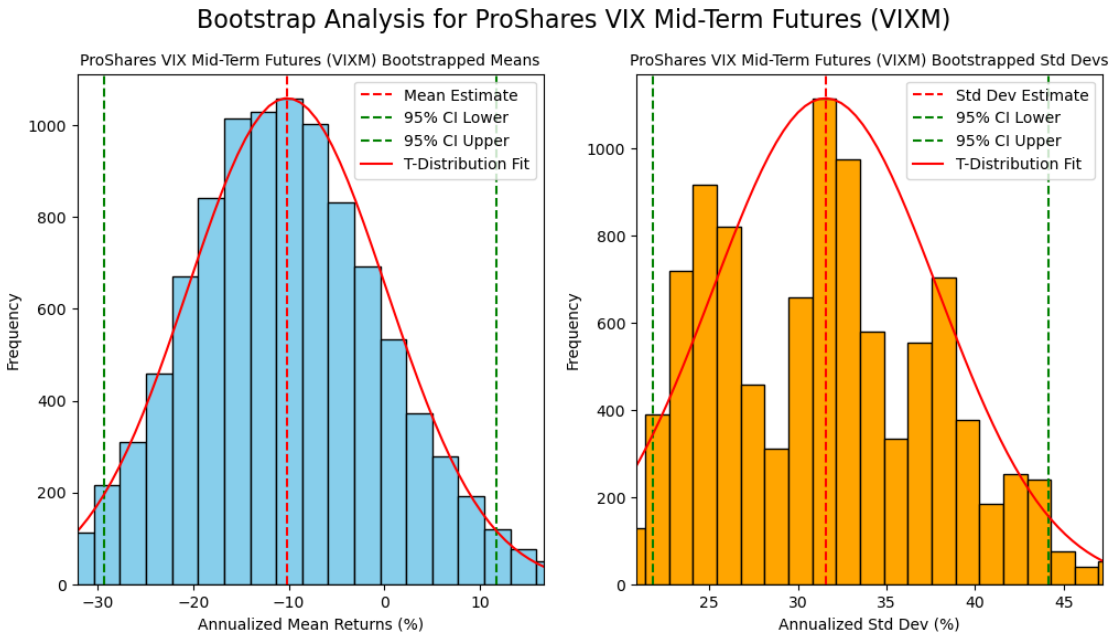


Bootstrap Analysis for AGF U.S. Market Neutral Anti-Beta (BTAL)




Bootstrap Analysis for AQR Managed Futures Strategy HV I (QMHIX)
AQR Managed Futures Strategy HV I (QMHIX) Bootstrapped Means
Mean Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Mean Returns (%)
AQR Managed Futures Strategy HV I (QMHIX) Bootstrapped Std Devs
Std Dev Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Std Dev (%)


Bootstrap Analysis for Invesco DB US Dollar Bullish (UUP)
Invesco DB US Dollar Bullish (UUP) Bootstrapped Means
Mean Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Mean Returns (%)
Invesco DB US Dollar Bullish (UUP) Bootstrapped Std Devs
Std Dev Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Std Dev (%)



20 Rolling 12-Month Correlations of Each Asset with VASIX (Not an Exhibit in the Report)

[31]:
```
import os
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from PIL import Image
from IPython.display import display
from scipy import stats

# ------------------------------ #
#         Load the Data          #
# ------------------------------ #

# Define the file path
file_path = 'Opportunity_Set.xlsx'

# Load the dataset
data = pd.read_excel(file_path)

# Ensure 'Date' is set as the index
if 'Date' in data.columns:
    data = data.set_index('Date')
```

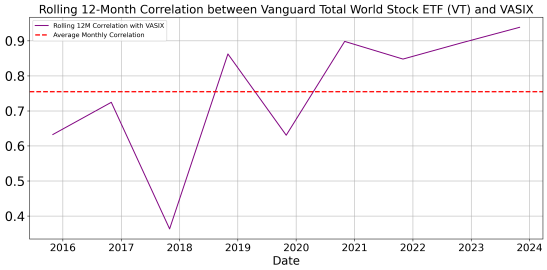
```
# Sort the index to ensure chronological order
data = data.sort_index()

# ------------------------------ #
#        Validate Benchmark      #
# ------------------------------ #

# Define the benchmark's full column name and its display label
benchmark_column = "Vanguard LifeStrategy Income Fund (VASIX)"
benchmark_label = 'VASIX'

# Check if the benchmark exists in the data
if benchmark_column not in data.columns:
    raise ValueError(f"Benchmark '{benchmark_label}' with column name␣
 ↪'{benchmark_column}' not found in the dataset columns.")

# Extract the benchmark returns
benchmark_returns = data[benchmark_column].dropna()

# ------------------------------ #
#      Prepare Output Folder     #
# ------------------------------ #

# Define a descriptive output folder name
output_folder = "Rolling_12M_Correlation_vs_VASIX"

# Create the output folder if it doesn't exist
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# ------------------------------ #
#    Define Downsampling Function#
# ------------------------------ #

def downsample_rolling_metrics(rolling_metric, window_size):
    """
    Downsamples rolling metrics by selecting every 'window_size' observations,
    effectively creating non-overlapping windows.

    Parameters:
    - rolling_metric: pd.DataFrame or pd.Series, the rolling metric to␣
 ↪downsample
    - window_size: int, the size of the rolling window

    Returns:
    - downsampled_metric: pd.DataFrame or pd.Series
    """
```

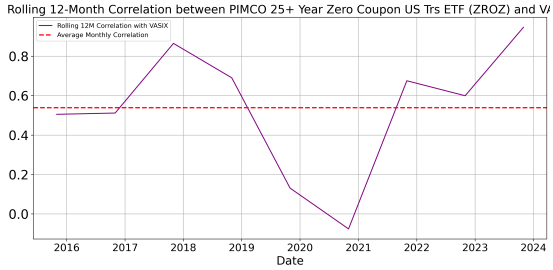
```
    # Drop NaN values resulting from rolling window
    rolling_metric = rolling_metric.dropna()

    # Select every 'window_size' observation to ensure non-overlapping
    downsampled_metric = rolling_metric.iloc[::window_size]

    return downsampled_metric

# ------------------------------ #
#   Calculate Rolling Correlation #
# ------------------------------ #

# Define rolling window size
window_size = 12  # 12 months

# Identify all assets excluding the benchmark
assets = [col for col in data.columns if col != benchmark_column]

# Initialize a dictionary to store rolling correlations
rolling_correlations = pd.DataFrame()

# Loop through each asset to compute rolling correlation with VASIX
for asset in assets:
    # Extract asset returns and benchmark returns
    asset_returns = data[asset].dropna()
    aligned_returns = pd.concat([asset_returns, benchmark_returns], axis=1).
 ↪dropna()

    # Compute rolling 12-month correlation
    rolling_corr = aligned_returns[asset].rolling(window=window_size).
 ↪corr(aligned_returns[benchmark_column])

    # Store the rolling correlation in the DataFrame
    rolling_correlations[asset] = rolling_corr

# ------------------------------ #
#   Downsample to Reduce Autocorr #
# ------------------------------ #

# Downsample rolling correlations to non-overlapping windows
downsampled_correlations = downsample_rolling_metrics(rolling_correlations,␣
 ↪window_size)

# Drop any remaining NaN values after downsampling
downsampled_correlations = downsampled_correlations.dropna()

# ------------------------------ #
```

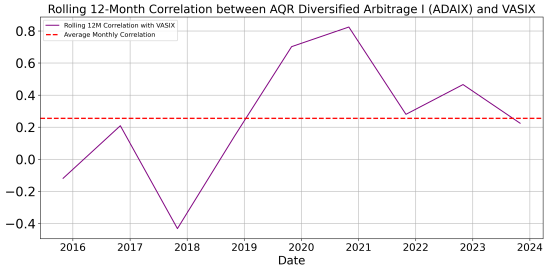
```
#    Define Plotting Function    #
# ------------------------------ #

def plot_rolling_correlation(corr_series, asset_name, benchmark_label,␣
 ↪output_folder):
    """
    Plots rolling 12-month correlation between an asset and VASIX.

    Parameters:
    - corr_series: pd.Series, the rolling correlation series
    - asset_name: str, name of the asset
    - benchmark_label: str, label for the benchmark (e.g., 'VASIX')
    - output_folder: str, path to the folder where the plot will be saved
    """
    plt.figure(figsize=(12, 6))
    plt.plot(corr_series.index, corr_series, color='purple', label=f'Rolling␣
 ↪12M Correlation with {benchmark_label}')

    # Calculate and plot the average monthly correlation over the full timespan
    avg_correlation = corr_series.mean()
    plt.axhline(y=avg_correlation, color='red', linestyle='--', linewidth=2,␣
 ↪label='Average Monthly Correlation')

    plt.xlabel("Date", fontsize=18)  # Doubled the font size of x-axis label
    #plt.ylabel("Rolling Correlation", fontsize=18)  # Doubled the font size of␣
 ↪y-axis label
    plt.xticks(fontsize=16)  # Doubled the font size of x-axis tick numbers
    plt.yticks(fontsize=20)  # Doubled the font size of y-axis tick numbers
    plt.title(f"Rolling 12-Month Correlation between {asset_name} and␣
 ↪{benchmark_label}", fontsize=18)
    plt.legend(loc='upper left')
    plt.grid(True)
    plt.tight_layout()

    # Define a clear and descriptive filename
    filename = f"{asset_name}_Rolling12M_Correlation_with_{benchmark_label}.png"
    plt.savefig(os.path.join(output_folder, filename), bbox_inches='tight',␣
 ↪dpi=300)
    plt.close()

# ------------------------------ #
#     Generate Individual Plots  #
# ------------------------------ #

# Loop through each asset and generate correlation plots
for asset in assets:
```

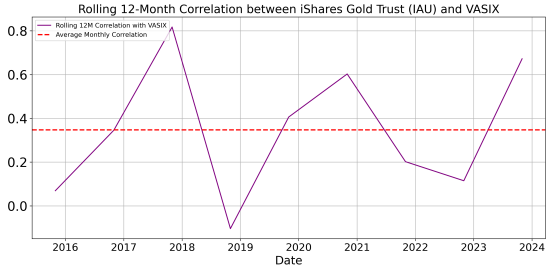
```
    # Extract the downsampled rolling correlation series for the asset
    corr_series = downsampled_correlations[asset].dropna()

    # Check if there are sufficient data points to plot
    if corr_series.empty:
        print(f"Skipping {asset}: Insufficient data after downsampling.")
        continue

    # Plot and save the rolling correlation
    plot_rolling_correlation(corr_series, asset, benchmark_label, output_folder)

# ------------------------------ #
#     Combine Individual Plots   #
# ------------------------------ #

# Gather all individual plot filenames
plot_filenames = [f"{asset}_Rolling12M_Correlation_with_{benchmark_label}.png"␣
 ↪for asset in assets]

# Verify that plot files exist
existing_plot_files = [os.path.join(output_folder, fname) for fname in␣
 ↪plot_filenames if os.path.exists(os.path.join(output_folder, fname))]

if not existing_plot_files:
    raise FileNotFoundError("No individual plot files found to combine.")

# Open all existing plot images
metrics_images = [Image.open(fname) for fname in existing_plot_files]

# Get dimensions from the first image
width, height = metrics_images[0].size

# Define the layout for the combined image (e.g., 2 columns)
num_columns = 2
num_rows = (len(metrics_images) + num_columns - 1) // num_columns
combined_width = width * num_columns
combined_height = height * num_rows

# Create a blank canvas for the combined image
combined_metrics_image = Image.new('RGB', (combined_width, combined_height),␣
 ↪(255, 255, 255))

# Paste each individual image into the combined canvas
for idx, image in enumerate(metrics_images):
    x_offset = (idx % num_columns) * width
    y_offset = (idx // num_columns) * height
    combined_metrics_image.paste(image, (x_offset, y_offset))
```

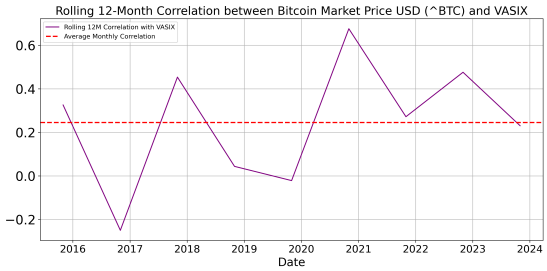
```
# Save and display the combined image
combined_image_filename =␣
 ↪f"Combined_Rolling12M_Correlation_vs_{benchmark_label}.png"
combined_metrics_image_path = os.path.join(output_folder,␣
 ↪combined_image_filename)
combined_metrics_image.save(combined_metrics_image_path, format='PNG')
display(combined_metrics_image)
```

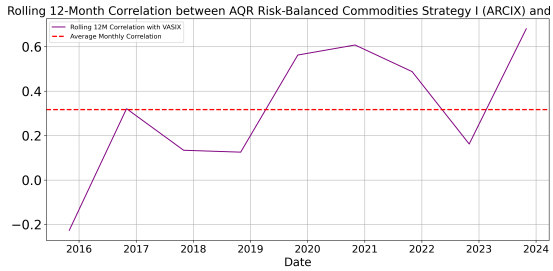

Rolling 12-Month Correlation between Vanguard Total World Stock ETF (VT) and VASIX
Rolling 12M Correlation with VASIX
Average Monthly Correlation
Date

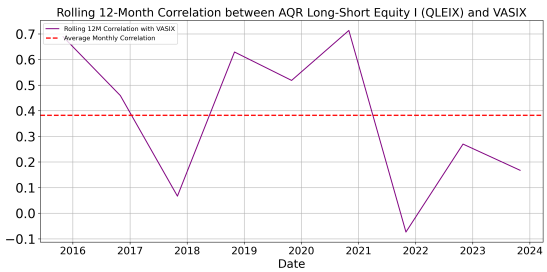

Rolling 12-Month Correlation between PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ) and VA
Rolling 12M Correlation with VASIX
Average Monthly Correlation
Date

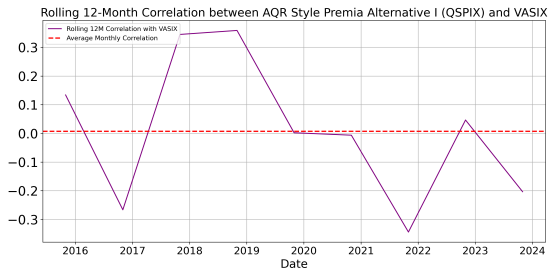

Rolling 12-Month Correlation between AQR Diversified Arbitrage I (ADAIX) and VASIX
Rolling 12M Correlation with VASIX
Average Monthly Correlation
Date

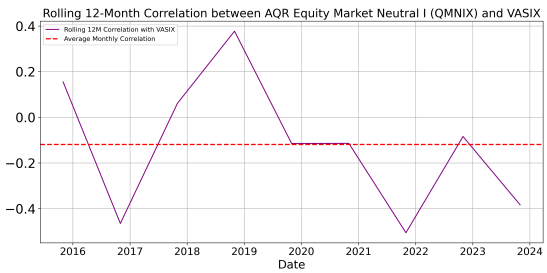

Rolling 12-Month Correlation between iShares Gold Trust (IAU) and VASIX
Rolling 12M Correlation with VASIX
Average Monthly Correlation
Date

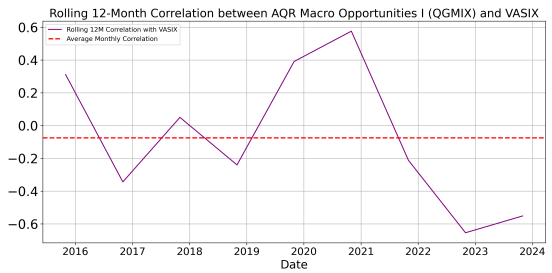

Rolling 12-Month Correlation between Bitcoin Market Price USD (^BTC) and VASIX
Rolling 12M Correlation with VASIX
Average Monthly Correlation
Date

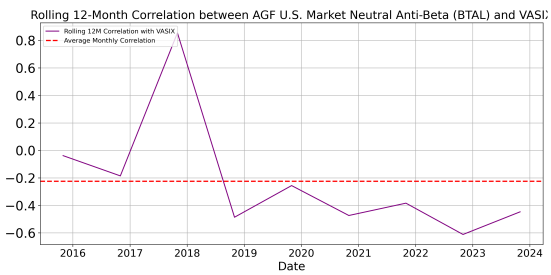

Rolling 12-Month Correlation between AQR Risk-Balanced Commodities Strategy I (ARCIX) and
Rolling 12M Correlation with VASIX
Average Monthly Correlation
Date

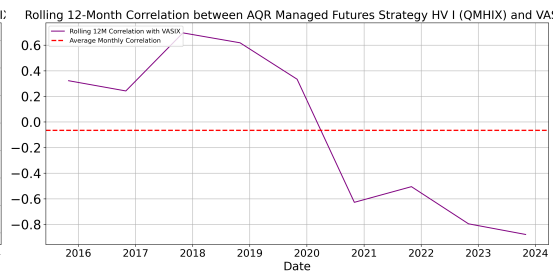

Rolling 12-Month Correlation between AQR Long-Short Equity I (QLEIX) and VASIX
Rolling 12M Correlation with VASIX
Average Monthly Correlation
Date

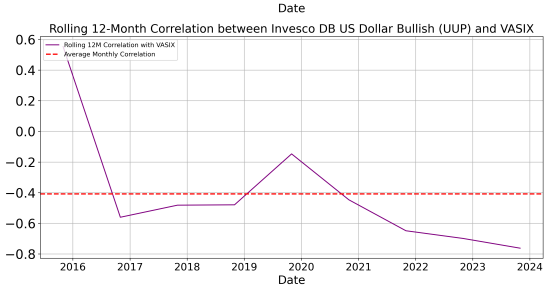

Rolling 12-Month Correlation between AQR Style Premia Alternative I (QSPIX) and VASIX
Rolling 12M Correlation with VASIX
Average Monthly Correlation
Date

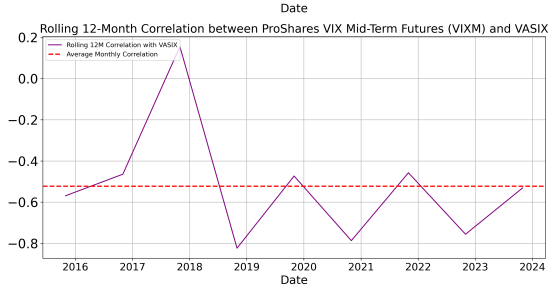

Rolling 12-Month Correlation between AQR Equity Market Neutral I (QMNIX) and VASIX
Rolling 12M Correlation with VASIX
Average Monthly Correlation
Date


Rolling 12-Month Correlation between AQR Macro Opportunities I (QGMIX) and VASIX
Rolling 12M Correlation with VASIX
Average Monthly Correlation
Date


Rolling 12-Month Correlation between AGF U.S. Market Neutral Anti-Beta (BTAL) and VASI
Rolling 12M Correlation with VASIX
Average Monthly Correlation
Date


Rolling 12-Month Correlation between AQR Managed Futures Strategy HV I (QMHIX) and VA
Rolling 12M Correlation with VASIX
Average Monthly Correlation
Date


Rolling 12-Month Correlation between Invesco DB US Dollar Bullish (UUP) and VASIX
Rolling 12M Correlation with VASIX
Average Monthly Correlation
Date


Rolling 12-Month Correlation between ProShares VIX Mid-Term Futures (VIXM) and VASIX
Rolling 12M Correlation with VASIX
Average Monthly Correlation
Date

21 Rolling 12-Month Covariance of Each Asset with VASIX (Not an Exhibit in the Report)

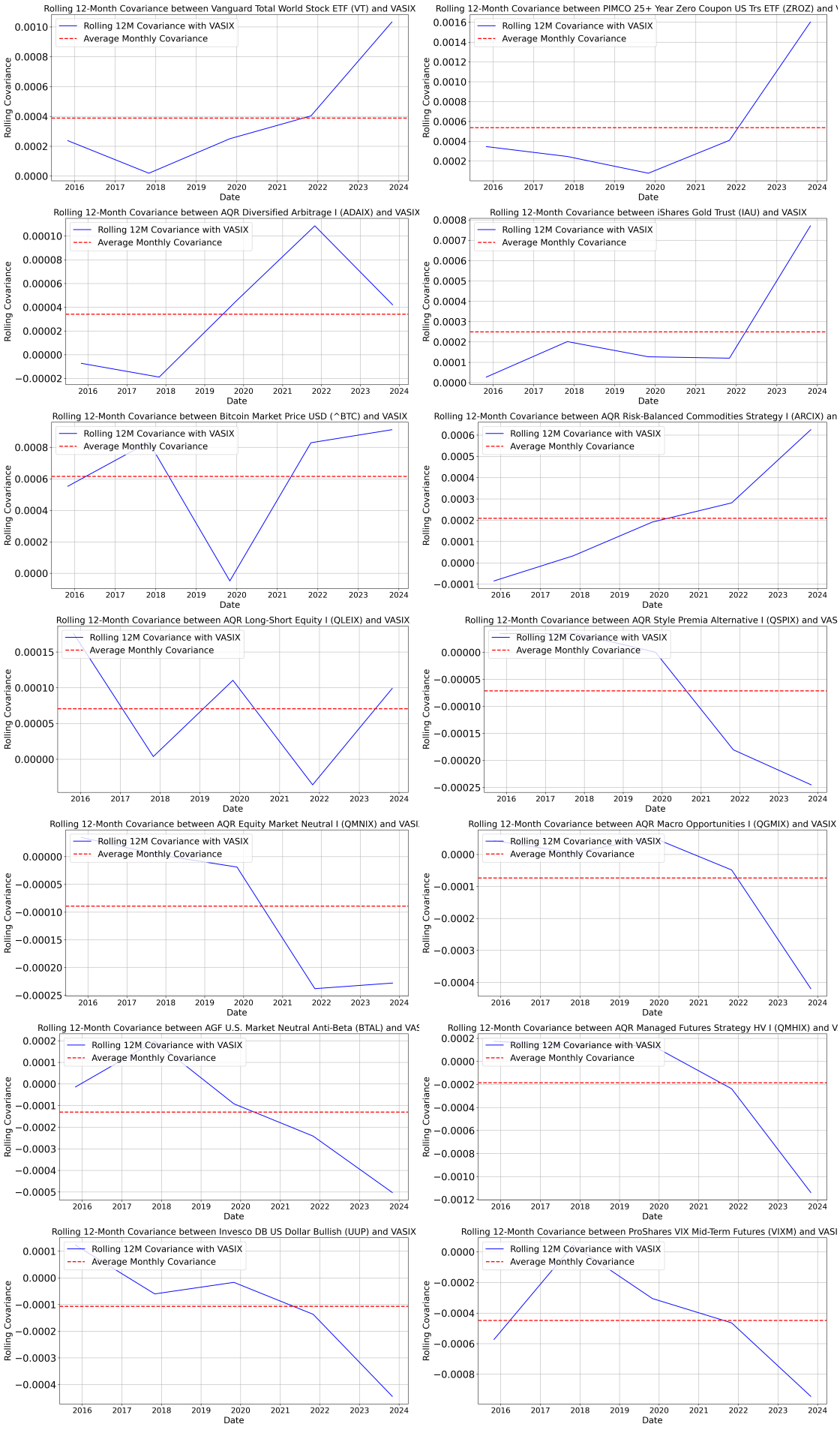
[32]:
```
import os
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from PIL import Image
from IPython.display import display
from scipy import stats

# ------------------------------ #
#         Load the Data          #
# ------------------------------ #

# Define the file path
file_path = 'Opportunity_Set.xlsx'

# Load the dataset
data = pd.read_excel(file_path)

# Ensure 'Date' is set as the index
if 'Date' in data.columns:
    data = data.set_index('Date')

# Sort the index to ensure chronological order
data = data.sort_index()

# ------------------------------ #
#        Validate Benchmark      #
# ------------------------------ #

# Define the benchmark's full column name and its display label
benchmark_column = "Vanguard LifeStrategy Income Fund (VASIX)"
benchmark_label = 'VASIX'

# Check if the benchmark exists in the data
if benchmark_column not in data.columns:
    raise ValueError(f"Benchmark '{benchmark_label}' with column name␣
 ↪'{benchmark_column}' not found in the dataset columns.")

# Extract the benchmark returns
benchmark_returns = data[benchmark_column].dropna()
```

```
# ------------------------------ #
#      Prepare Output Folder     #
# ------------------------------ #

# Define a descriptive output folder name
output_folder = "Rolling_12M_Covariance_vs_VASIX"

# Create the output folder if it doesn't exist
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# ------------------------------ #
#    Define Downsampling Function#
# ------------------------------ #

def downsample_rolling_metrics(rolling_metric, window_size):
    """
    Downsamples rolling metrics by selecting every 'window_size' observations,
    effectively creating non-overlapping windows.

    Parameters:
    - rolling_metric: pd.DataFrame or pd.Series, the rolling metric to␣
 ↪downsample
    - window_size: int, the size of the rolling window

    Returns:
    - downsampled_metric: pd.DataFrame or pd.Series
    """
    # Drop NaN values resulting from rolling window
    rolling_metric = rolling_metric.dropna()

    # Select every 'window_size' observation to ensure non-overlapping
    downsampled_metric = rolling_metric.iloc[::window_size]

    return downsampled_metric

# ------------------------------ #
#   Calculate Rolling Covariance #
# ------------------------------ #

# Define rolling window size
window_size = 12  # 12 months

# Identify all assets excluding the benchmark
assets = [col for col in data.columns if col != benchmark_column]
```

```
# Initialize a dictionary to store rolling covariances
rolling_covariances = pd.DataFrame()

# Loop through each asset to compute rolling covariance with VASIX
for asset in assets:
    # Extract asset returns and benchmark returns
    asset_returns = data[asset].dropna()
    aligned_returns = pd.concat([asset_returns, benchmark_returns], axis=1).
 ↪dropna()

    # Compute rolling 12-month covariance
    rolling_cov = aligned_returns[asset].rolling(window=window_size).
 ↪cov(aligned_returns[benchmark_column])
    rolling_cov = rolling_cov.iloc[1::2]  # Select every second row to get the␣
 ↪covariance values

    # Store the rolling covariance in the DataFrame
    rolling_covariances[asset] = rolling_cov

# ------------------------------ #
#   Downsample to Reduce Autocorr #
# ------------------------------ #

# Downsample rolling covariances to non-overlapping windows
downsampled_covariances = downsample_rolling_metrics(rolling_covariances,␣
 ↪window_size)

# Drop any remaining NaN values after downsampling
downsampled_covariances = downsampled_covariances.dropna()

# ------------------------------ #
#    Define Plotting Function    #
# ------------------------------ #

def plot_rolling_covariance(cov_series, asset_name, benchmark_label,␣
 ↪output_folder):
    """
    Plots rolling 12-month covariance between an asset and VASIX.

    Parameters:
    - cov_series: pd.Series, the rolling covariance series
    - asset_name: str, name of the asset
    - benchmark_label: str, label for the benchmark (e.g., 'VASIX')
    - output_folder: str, path to the folder where the plot will be saved
    """
    plt.figure(figsize=(12, 6))
```

```
    plt.plot(cov_series.index, cov_series, color='blue', label=f'Rolling 12M␣
↪Covariance with {benchmark_label}')

    # Calculate and plot the average monthly covariance over the full timespan
    avg_covariance = cov_series.mean()
    plt.axhline(y=avg_covariance, color='red', linestyle='--', linewidth=2,␣
↪label='Average Monthly Covariance')

    plt.xlabel("Date", fontsize=18)  # Doubled the font size of x-axis label
    plt.ylabel("Rolling Covariance", fontsize=18)  # Doubled the font size of␣
↪y-axis label
    plt.xticks(fontsize=16)  # Doubled the font size of x-axis tick numbers
    plt.yticks(fontsize=20)  # Doubled the font size of y-axis tick numbers
    plt.title(f"Rolling 12-Month Covariance between {asset_name} and␣
↪{benchmark_label}", fontsize=18)
    plt.legend(loc='upper left',fontsize=18)
    plt.grid(True)
    plt.tight_layout()

    # Define a clear and descriptive filename
    filename = f"{asset_name}_Rolling12M_Covariance_with_{benchmark_label}.png"
    plt.savefig(os.path.join(output_folder, filename), bbox_inches='tight',␣
↪dpi=300)
    plt.close()

# ------------------------------ #
#     Generate Individual Plots  #
# ------------------------------ #

# Loop through each asset and generate covariance plots
for asset in assets:
    # Extract the downsampled rolling covariance series for the asset
    cov_series = downsampled_covariances[asset].dropna()

    # Check if there are sufficient data points to plot
    if cov_series.empty:
        print(f"Skipping {asset}: Insufficient data after downsampling.")
        continue

    # Plot and save the rolling covariance
    plot_rolling_covariance(cov_series, asset, benchmark_label, output_folder)

# ------------------------------ #
#     Combine Individual Plots   #
# ------------------------------ #

# Gather all individual plot filenames
```

```
plot_filenames = [f"{asset}_Rolling12M_Covariance_with_{benchmark_label}.png"␣
 ↪for asset in assets]

# Verify that plot files exist
existing_plot_files = [os.path.join(output_folder, fname) for fname in␣
 ↪plot_filenames if os.path.exists(os.path.join(output_folder, fname))]

if not existing_plot_files:
    raise FileNotFoundError("No individual plot files found to combine.")

# Open all existing plot images
metrics_images = [Image.open(fname) for fname in existing_plot_files]

# Get dimensions from the first image
width, height = metrics_images[0].size

# Define the layout for the combined image (e.g., 2 columns)
num_columns = 2
num_rows = (len(metrics_images) + num_columns - 1) // num_columns
combined_width = width * num_columns
combined_height = height * num_rows

# Create a blank canvas for the combined image
combined_metrics_image = Image.new('RGB', (combined_width, combined_height),␣
 ↪(255, 255, 255))

# Paste each individual image into the combined canvas
for idx, image in enumerate(metrics_images):
    x_offset = (idx % num_columns) * width
    y_offset = (idx // num_columns) * height
    combined_metrics_image.paste(image, (x_offset, y_offset))

# Save and display the combined image
combined_image_filename = f"Combined_Rolling12M_Covariance_vs_{benchmark_label}.
 ↪png"
combined_metrics_image_path = os.path.join(output_folder,␣
 ↪combined_image_filename)
combined_metrics_image.save(combined_metrics_image_path, format='PNG')
display(combined_metrics_image)
```


Rolling 12-Month Covariance between Vanguard Total World Stock ETF (VT) and VASIX
Rolling 12-Month Covariance between PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ) and
Rolling 12-Month Covariance between AQR Diversified Arbitrage I (ADAIX) and VASIX
Rolling 12-Month Covariance between iShares Gold Trust (IAU) and VASIX
Rolling 12-Month Covariance between Bitcoin Market Price USD (^BTC) and VASIX
Rolling 12-Month Covariance between AQR Risk-Balanced Commodities Strategy I (ARCIX) an
Rolling 12-Month Covariance between AQR Long-Short Equity I (QLEIX) and VASIX
Rolling 12-Month Covariance between AQR Style Premia Alternative I (QSPIX) and VAS
Rolling 12-Month Covariance between AQR Equity Market Neutral I (QMNIX) and VASI
Rolling 12-Month Covariance between AQR Macro Opportunities I (QGMIX) and VASIX
Rolling 12-Month Covariance between AGF U.S. Market Neutral Anti-Beta (BTAL) and VAS
Rolling 12-Month Covariance between AQR Managed Futures Strategy HV I (QMHIX) and V
Rolling 12-Month Covariance between Invesco DB US Dollar Bullish (UUP) and VASIX
Rolling 12-Month Covariance between ProShares VIX Mid-Term Futures (VIXM) and VASI
Rolling 12M Covariance with VASIX
Average Monthly Covariance
Rolling Covariance
Date

22 Rolling 12-Month Beta Coefficient of Each Asset with VASIX (Not an Exhibit in the Report)

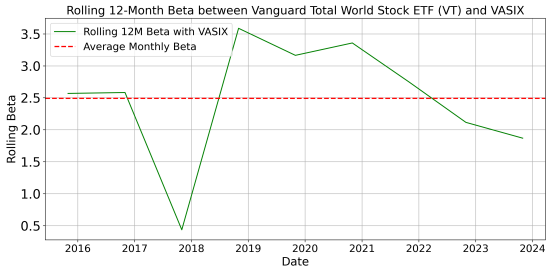
[33]:
```
import os
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from PIL import Image
from IPython.display import display
from scipy import stats

# ------------------------------ #
#         Load the Data          #
# ------------------------------ #

# Define the file path
file_path = 'Opportunity_Set.xlsx'

# Load the dataset
data = pd.read_excel(file_path)

# Ensure 'Date' is set as the index
if 'Date' in data.columns:
    data = data.set_index('Date')

# Sort the index to ensure chronological order
data = data.sort_index()

# ------------------------------ #
#        Validate Benchmark      #
# ------------------------------ #

# Define the benchmark's full column name and its display label
benchmark_column = "Vanguard LifeStrategy Income Fund (VASIX)"
benchmark_label = 'VASIX'

# Check if the benchmark exists in the data
if benchmark_column not in data.columns:
    raise ValueError(f"Benchmark '{benchmark_label}' with column name␣
 ↪'{benchmark_column}' not found in the dataset columns.")

# Extract the benchmark returns
benchmark_returns = data[benchmark_column].dropna()
```

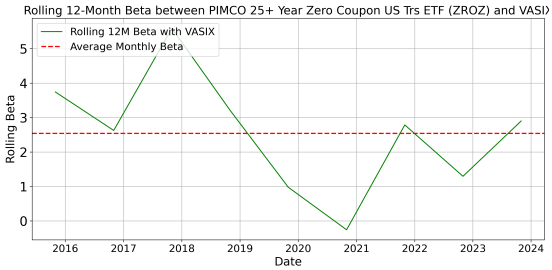
```
# ------------------------------ #
#      Prepare Output Folder     #
# ------------------------------ #

# Define a descriptive output folder name
output_folder = "Rolling_12M_Beta_vs_VASIX"

# Create the output folder if it doesn't exist
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# ------------------------------ #
#    Define Downsampling Function#
# ------------------------------ #

def downsample_rolling_metrics(rolling_metric, window_size):
    """
    Downsamples rolling metrics by selecting every 'window_size' observations,
    effectively creating non-overlapping windows.

    Parameters:
    - rolling_metric: pd.DataFrame or pd.Series, the rolling metric to␣
 ↪downsample
    - window_size: int, the size of the rolling window

    Returns:
    - downsampled_metric: pd.DataFrame or pd.Series
    """
    # Drop NaN values resulting from rolling window
    rolling_metric = rolling_metric.dropna()

    # Select every 'window_size' observation to ensure non-overlapping
    downsampled_metric = rolling_metric.iloc[::window_size]

    return downsampled_metric

# ------------------------------ #
#   Calculate Rolling Beta Coefficients #
# ------------------------------ #

# Define rolling window size
window_size = 12  # 12 months

# Identify all assets excluding the benchmark
assets = [col for col in data.columns if col != benchmark_column]
```

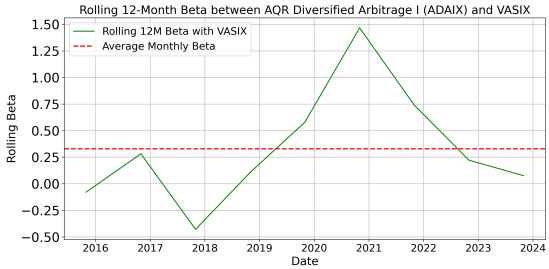
```
# Initialize a dictionary to store rolling betas
rolling_betas = pd.DataFrame()

# Loop through each asset to compute rolling beta with VASIX
for asset in assets:
    # Extract asset returns and benchmark returns
    asset_returns = data[asset].dropna()
    aligned_returns = pd.concat([asset_returns, benchmark_returns], axis=1).
 ↪dropna()

    # Compute rolling 12-month beta
    rolling_beta = aligned_returns[asset].rolling(window=window_size).apply(
        lambda x: stats.linregress(aligned_returns[benchmark_column].loc[x.
 ↪index], x)[0], raw=False
    )

    # Store the rolling beta in the DataFrame
    rolling_betas[asset] = rolling_beta

# ------------------------------ #
#   Downsample to Reduce Autocorr #
# ------------------------------ #

# Downsample rolling betas to non-overlapping windows
downsampled_betas = downsample_rolling_metrics(rolling_betas, window_size)

# Drop any remaining NaN values after downsampling
downsampled_betas = downsampled_betas.dropna()

# ------------------------------ #
#    Define Plotting Function    #
# ------------------------------ #

def plot_rolling_beta(beta_series, asset_name, benchmark_label, output_folder):
    """
    Plots rolling 12-month beta between an asset and VASIX.

    Parameters:
    - beta_series: pd.Series, the rolling beta series
    - asset_name: str, name of the asset
    - benchmark_label: str, label for the benchmark (e.g., 'VASIX')
    - output_folder: str, path to the folder where the plot will be saved
    """
    plt.figure(figsize=(12, 6))
    plt.plot(beta_series.index, beta_series, color='green', label=f'Rolling 12M␣
 ↪Beta with {benchmark_label}')
```

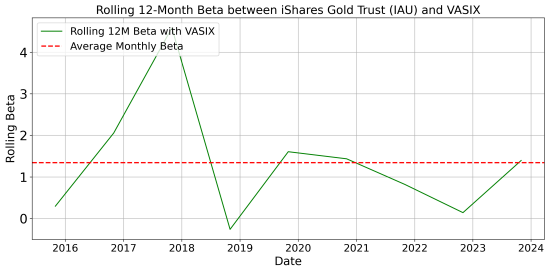
```
    # Calculate and plot the average monthly beta over the full timespan
    avg_beta = beta_series.mean()
    plt.axhline(y=avg_beta, color='red', linestyle='--', linewidth=2,␣
 ↪label='Average Monthly Beta')

    plt.xlabel("Date", fontsize=18)  # Doubled the font size of x-axis label
    plt.ylabel("Rolling Beta", fontsize=18)  # Doubled the font size of y-axis␣
 ↪label
    plt.xticks(fontsize=16)  # Doubled the font size of x-axis tick numbers
    plt.yticks(fontsize=20)  # Doubled the font size of y-axis tick numbers
    plt.title(f"Rolling 12-Month Beta between {asset_name} and␣
 ↪{benchmark_label}", fontsize=18)
    plt.legend(loc='upper left', fontsize=16)
    plt.grid(True)
    plt.tight_layout()

    # Define a clear and descriptive filename
    filename = f"{asset_name}_Rolling12M_Beta_with_{benchmark_label}.png"
    plt.savefig(os.path.join(output_folder, filename), bbox_inches='tight',␣
 ↪dpi=300)
    plt.close()

# ------------------------------ #
#     Generate Individual Plots  #
# ------------------------------ #

# Loop through each asset and generate beta plots
for asset in assets:
    # Extract the downsampled rolling beta series for the asset
    beta_series = downsampled_betas[asset].dropna()

    # Check if there are sufficient data points to plot
    if beta_series.empty:
        print(f"Skipping {asset}: Insufficient data after downsampling.")
        continue

    # Plot and save the rolling beta
    plot_rolling_beta(beta_series, asset, benchmark_label, output_folder)

# ------------------------------ #
#     Combine Individual Plots   #
# ------------------------------ #

# Gather all individual plot filenames
plot_filenames = [f"{asset}_Rolling12M_Beta_with_{benchmark_label}.png" for␣
 ↪asset in assets]
```

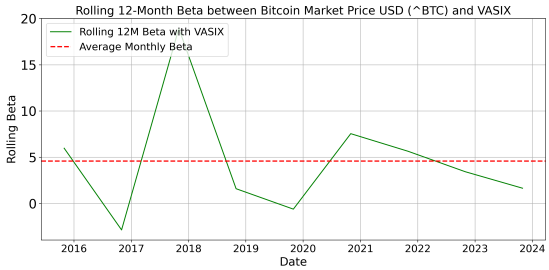
```
# Verify that plot files exist
existing_plot_files = [os.path.join(output_folder, fname) for fname in␣
 ↪plot_filenames if os.path.exists(os.path.join(output_folder, fname))]

if not existing_plot_files:
    raise FileNotFoundError("No individual plot files found to combine.")

# Open all existing plot images
metrics_images = [Image.open(fname) for fname in existing_plot_files]

# Get dimensions from the first image
width, height = metrics_images[0].size

# Define the layout for the combined image (e.g., 2 columns)
num_columns = 2
num_rows = (len(metrics_images) + num_columns - 1) // num_columns
combined_width = width * num_columns
combined_height = height * num_rows

# Create a blank canvas for the combined image
combined_metrics_image = Image.new('RGB', (combined_width, combined_height),␣
 ↪(255, 255, 255))

# Paste each individual image into the combined canvas
for idx, image in enumerate(metrics_images):
    x_offset = (idx % num_columns) * width
    y_offset = (idx // num_columns) * height
    combined_metrics_image.paste(image, (x_offset, y_offset))

# Save and display the combined image
combined_image_filename = f"Combined_Rolling12M_Beta_vs_{benchmark_label}.png"
combined_metrics_image_path = os.path.join(output_folder,␣
 ↪combined_image_filename)
combined_metrics_image.save(combined_metrics_image_path, format='PNG')
display(combined_metrics_image)
```

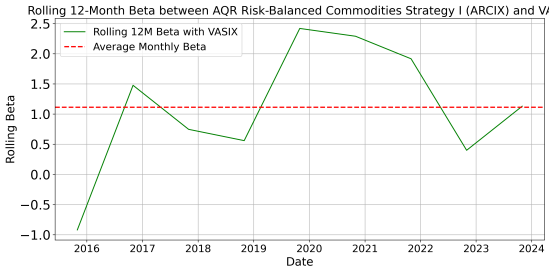

Rolling 12-Month Beta between Vanguard Total World Stock ETF (VT) and VASIX
Rolling 12M Beta with VASIX
Average Monthly Beta
Rolling Beta
Date

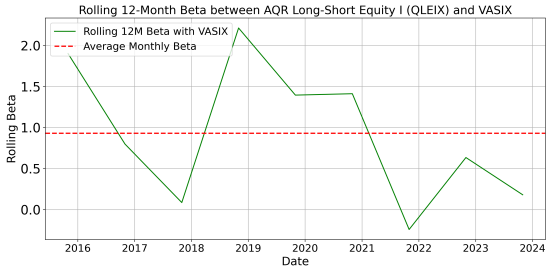

Rolling 12-Month Beta between PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ) and VASI
Rolling 12M Beta with VASIX
Average Monthly Beta
Rolling Beta
Date

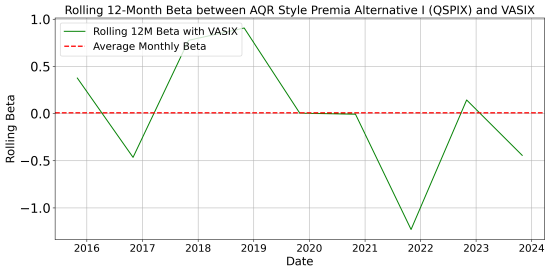

Rolling 12-Month Beta between AQR Diversified Arbitrage I (ADAIX) and VASIX
Rolling 12M Beta with VASIX
Average Monthly Beta
Rolling Beta
Date

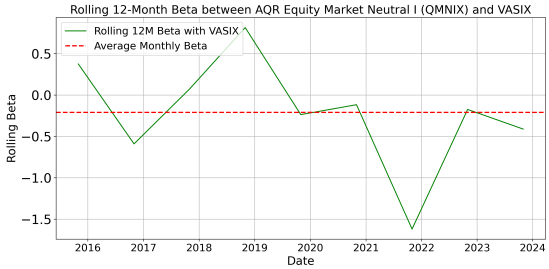

Rolling 12-Month Beta between iShares Gold Trust (IAU) and VASIX
Rolling 12M Beta with VASIX
Average Monthly Beta
Rolling Beta
Date

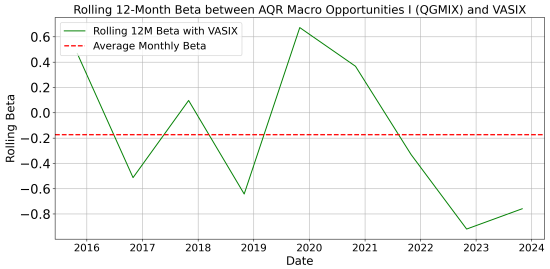

Rolling 12-Month Beta between Bitcoin Market Price USD (^BTC) and VASIX
Rolling 12M Beta with VASIX
Average Monthly Beta
Rolling Beta
Date

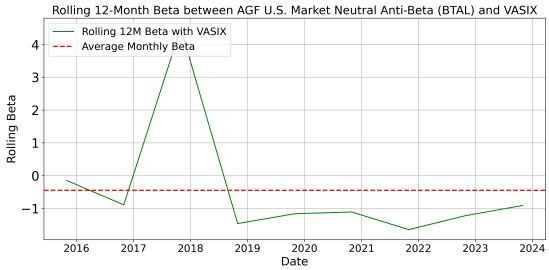

Rolling 12-Month Beta between AQR Risk-Balanced Commodities Strategy I (ARCIX) and VA
Rolling 12M Beta with VASIX
Average Monthly Beta
Rolling Beta
Date

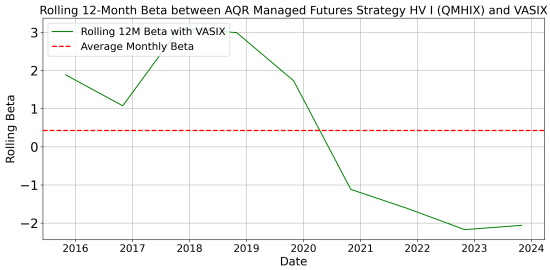

Rolling 12-Month Beta between AQR Long-Short Equity I (QLEIX) and VASIX
Rolling 12M Beta with VASIX
Average Monthly Beta
Rolling Beta
Date

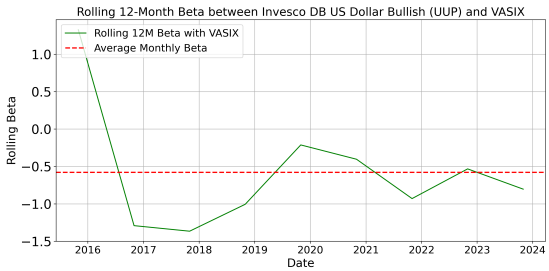

Rolling 12-Month Beta between AQR Style Premia Alternative I (QSPIX) and VASIX
Rolling 12M Beta with VASIX
Average Monthly Beta
Rolling Beta
Date

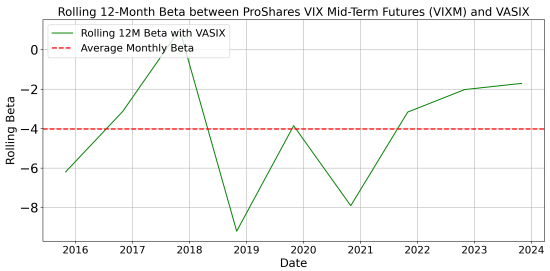

Rolling 12-Month Beta between AQR Equity Market Neutral I (QMNIX) and VASIX
Rolling 12M Beta with VASIX
Average Monthly Beta
Rolling Beta
Date


Rolling 12-Month Beta between AQR Macro Opportunities I (QGMIX) and VASIX
Rolling 12M Beta with VASIX
Average Monthly Beta
Rolling Beta
Date


Rolling 12-Month Beta between AGF U.S. Market Neutral Anti-Beta (BTAL) and VASIX
Rolling 12M Beta with VASIX
Average Monthly Beta
Rolling Beta
Date


Rolling 12-Month Beta between AQR Managed Futures Strategy HV I (QMHIX) and VASIX
Rolling 12M Beta with VASIX
Average Monthly Beta
Rolling Beta
Date


Rolling 12-Month Beta between Invesco DB US Dollar Bullish (UUP) and VASIX
Rolling 12M Beta with VASIX
Average Monthly Beta
Rolling Beta
Date


Rolling 12-Month Beta between ProShares VIX Mid-Term Futures (VIXM) and VASIX
Rolling 12M Beta with VASIX
Average Monthly Beta
Rolling Beta
Date

23 Anderson-Darling, Shapiro-Wilk, D'Agostino K-squared, and Jarque-Bera tests (Not an Exhibit in Report)

[34]:
```
import numpy as np
import pandas as pd
from scipy.stats import anderson, shapiro, normaltest, jarque_bera
import os
import dataframe_image as dfi

# Load the data
file_path = 'Opportunity_Set.xlsx'
data = pd.read_excel(file_path)

# Ensure the returns data has a DateTimeIndex
data['Date'] = pd.to_datetime(data['Date'])  # Convert 'Date' column to datetime
data.set_index('Date', inplace=True)  # Set 'Date' as the index

# Create an empty list to store the results
distribution_suggestions = []

# Define the function to evaluate distributions
def evaluate_distribution(ad_stat, ad_critical_values, sw_pvalue,␣
 ↪dagostino_pvalue, jb_pvalue):
    """
    Function to evaluate which distribution is likely the best fit based on␣
 ↪test results.

    Parameters:
    - ad_stat: Anderson-Darling test statistic
    - ad_critical_values: Anderson-Darling critical values
    - sw_pvalue: Shapiro-Wilk p-value
    - dagostino_pvalue: D'Agostino K-squared test p-value
    - jb_pvalue: Jarque-Bera test p-value

    Returns:
    - String indicating the likely best distribution match.
    """
    # Start by checking Anderson-Darling Test
    if ad_stat > ad_critical_values[2]:  # Using a common significance level␣
 ↪threshold
        # Anderson-Darling rejects normality
        if sw_pvalue < 0.05 and dagostino_pvalue < 0.05 and jb_pvalue < 0.05:
```

```
            return "Likely a distribution with heavy tails or skewness␣
 ↪(t-distribution, skew-normal)"
        elif jb_pvalue > 0.05:
            return "Data may have moderate skewness or kurtosis. Consider␣
 ↪Skew-Normal distribution."
        else:
            return "Normal distribution may not be appropriate. Consider␣
 ↪alternatives like t-distribution."
    else:
        # Anderson-Darling does not reject normality
        if sw_pvalue > 0.05 and dagostino_pvalue > 0.05 and jb_pvalue > 0.05:
            return "Normal distribution is a reasonable fit."
        else:
            return "Data has some deviation from normality, possibly mild␣
 ↪skewness or kurtosis."

# Iterate over each asset to perform normality tests and evaluate distributions
for asset in data.columns:
    asset_returns = data[asset].dropna()

    # Anderson-Darling Test
    ad_test = anderson(asset_returns, dist='norm')
    ad_stat = round(ad_test.statistic, 2)
    ad_crit_values = [round(cv, 2) for cv in ad_test.critical_values]

    # Shapiro-Wilk Test
    shapiro_test = shapiro(asset_returns)
    shapiro_pvalue = round(shapiro_test.pvalue, 4)

    # D'Agostino's K-squared Test (aka normaltest)
    dagostino_test = normaltest(asset_returns)
    dagostino_pvalue = round(dagostino_test.pvalue, 4)

    # Jarque-Bera Test
    jb_test = jarque_bera(asset_returns)
    jb_pvalue = round(jb_test.pvalue, 4)

    # Evaluate the distribution
    distribution_suggestion = evaluate_distribution(ad_stat, ad_crit_values,␣
 ↪shapiro_pvalue, dagostino_pvalue, jb_pvalue)

    # Append the results to the list
    distribution_suggestions.append({
        "Asset": asset,
        "Anderson-Darling Statistic": ad_stat,
        "Anderson-Darling Critical Values": ad_crit_values,
        "Shapiro-Wilk p-value": shapiro_pvalue,
```

```
        "D'Agostino K-squared p-value": dagostino_pvalue,
        "Jarque-Bera p-value": jb_pvalue,
        "Suggested Distribution": distribution_suggestion
    })

# Convert the results to a DataFrame
distribution_suggestions_df = pd.DataFrame(distribution_suggestions)

# Style the distribution suggestions table
styled_distribution_table = distribution_suggestions_df.style.set_properties(
    **{'text-align': 'center'}
).set_table_styles([
    {'selector': 'th', 'props': [('text-align', 'center')]},
    {'selector': 'td, th', 'props': [('border', '1px solid black')]},
    {'selector': 'td.col0', 'props': [('text-align', 'left'), ('white-space',␣
 ↪'nowrap')]},  # Asset column no wrapping
]).set_caption("Distribution Evaluation for All Assets")

# Output folder setup for distribution suggestions
output_folder = 'DISTRIBUTION_SUGGESTIONS'
os.makedirs(output_folder, exist_ok=True)
output_file_path = os.path.join(output_folder, "distribution_suggestions_table.
 ↪png")

# Save the styled table as an image
dfi.export(styled_distribution_table, output_file_path)

# Display the styled table in Jupyter Notebook
styled_distribution_table
```

[34]: <pandas.io.formats.style.Styler at 0x343678e30>

24 Monte Carlo Simulations Based on T-Distributions (36-Mo, 10,000 iterations) (Tables 8, 9, 10)

[12]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import dataframe_image as dfi
from scipy.stats import t
from IPython.display import display

# Constants
PERCENTILES = [10, 50, 90]
COLORS = ['purple', 'blue', 'green']
```

```
LABELS = [f'{p}th Percentile' + (' (Median)' if p == 50 else '') for p in␣
 ↪PERCENTILES]
MONTHS = 36  # Number of months to simulate
INITIAL_BALANCE = 10000
NUM_SIMULATIONS = 10000
SEED = 42
DF_TDIST = 5  # Degrees of freedom for t-distribution

def style_summary_table(df, caption):
    """
    Styles the summary DataFrame for export and display.

    Parameters:
    - df (pd.DataFrame): The DataFrame to style.
    - caption (str): The caption for the styled table.

    Returns:
    - Styler object: The styled DataFrame.
    """
    return df.style.set_properties(**{'text-align': 'center'}) \
        .set_table_styles([
            {'selector': 'th', 'props': [('text-align', 'center')]},
            {'selector': 'td, th', 'props': [('border', '1px solid black')]},
            {'selector': 'td.col0', 'props': [('text-align', 'left'),␣
 ↪('white-space', 'nowrap')]},
            {'selector': 'td', 'props': [('padding-top', '8px'),␣
 ↪('padding-bottom', '8px')]}
        ]) \
        .format({
            "Portfolio End Balance ($)": "{:,.0f}",
            "Annual Compounded Return (%)": "{:.2f}",
            "Annualized Volatility (%)": "{:.2f}",
            "Maximum Drawdown (%)": "{:.2f}"
        }) \
        .set_caption(caption)

def plot_percentiles(cumulative_returns, asset_name, output_folder):
    """
    Plots the specified percentiles of cumulative returns.

    Parameters:
    - cumulative_returns (np.ndarray): Array of cumulative returns.
    - asset_name (str): Name of the asset.
    - output_folder (str): Directory to save the plot.
    """
    plt.figure(figsize=(10, 6), dpi=100)
    for p, color, label in zip(PERCENTILES, COLORS, LABELS):
```

```
        plt.plot(
            np.arange(0, MONTHS + 1),
            np.percentile(cumulative_returns, p, axis=0),
            label=label,
            color=color
        )
    plt.title(f"Monte Carlo Simulation of {asset_name} Over {MONTHS//12} Years")
    plt.xlabel("Months")
    plt.xticks(np.arange(0, MONTHS + 1, 6))
    plt.ylabel("Cumulative Return ($)")
    plt.legend(loc='upper left')
    plt.grid(True)

    plot_path = os.path.join(output_folder,␣
 ↪f"{asset_name}_TDIST_monte_carlo_simulation.png")
    plt.savefig(plot_path, bbox_inches='tight')
    plt.close()

def plot_combined_assets(cumulative_returns_dict, assets, output_folder):
    """
    Plots combined percentiles for all assets in a grid layout.

    Parameters:
    - cumulative_returns_dict (dict): Dictionary mapping asset names to their␣
 ↪cumulative returns.
    - assets (list): List of asset names.
    - output_folder (str): Directory to save the combined plot.
    """
    num_assets = len(assets)
    num_cols = 2
    num_rows = (num_assets + num_cols - 1) // num_cols if num_assets > 0 else 1
    fig, axs = plt.subplots(num_rows, num_cols, figsize=(15, 5 * num_rows))
    axs = axs.flatten()

    for idx, asset in enumerate(assets):
        ax = axs[idx]
        cum_returns = cumulative_returns_dict[asset]
        for p, color, label in zip(PERCENTILES, COLORS, LABELS):
            ax.plot(
                np.arange(0, MONTHS + 1),
                np.percentile(cum_returns, p, axis=0),
                label=label,
                color=color
            )
        ax.set_title(f"{asset} Monte Carlo")
        ax.set_xlabel("Months")
        ax.set_xticks(np.arange(0, MONTHS + 1, 6))
```

```
        ax.set_ylabel("Cumulative Return ($)")
        ax.legend(loc='upper left')
        ax.grid(True)

    # Remove any empty subplots
    for idx in range(num_assets, num_rows * num_cols):
        fig.delaxes(axs[idx])

    plt.tight_layout(pad=5.0)
    combined_plot_path = os.path.join(output_folder,␣
↪"TDIST_combined_asset_plots.png")
    plt.savefig(combined_plot_path, dpi=300, bbox_inches='tight')
    plt.close()

def export_summary_table(styled_df, path):
    """
    Exports the styled DataFrame as a PNG image.

    Parameters:
    - styled_df (Styler): The styled DataFrame to export.
    - path (str): File path to save the image.
    """
    dfi.export(styled_df, path, dpi=300)

def run_monte_carlo_simulations(returns, asset_name):
    """
    Performs Monte Carlo simulation using t-distribution for a given asset.

    Parameters:
    - returns (pd.Series): Historical returns of the asset.
    - asset_name (str): Name of the asset.

    Returns:
    - cumulative_returns_with_zero (np.ndarray): Simulated cumulative returns␣
↪including initial balance.
    - summary (pd.DataFrame): Summary statistics for the asset.
    - styled_summary (Styler): Styled summary for display.
    """
    np.random.seed(SEED)
    mean_monthly, std_monthly = returns.mean(), returns.std()

    # Simulate monthly returns using t-distribution
    simulations = t.rvs(DF_TDIST, loc=mean_monthly, scale=std_monthly,␣
↪size=(NUM_SIMULATIONS, MONTHS))

    # Calculate cumulative returns
    cumulative_returns = np.cumprod(1 + simulations, axis=1) * INITIAL_BALANCE
```

```
    cumulative_returns_with_zero = np.hstack((np.full((NUM_SIMULATIONS, 1),␣
↪INITIAL_BALANCE), cumulative_returns))

    # Portfolio End Balances at specified percentiles
    end_balances = np.percentile(cumulative_returns_with_zero[:, -1],␣
↪PERCENTILES)
    annual_compounded_return = (end_balances / INITIAL_BALANCE) ** (12 /␣
↪MONTHS) - 1

    # Annualized Volatility Calculation based on simulated monthly returns
    monthly_volatility = np.std(simulations, axis=1)
    annualized_volatility_percentiles = np.percentile(monthly_volatility * np.
↪sqrt(12), PERCENTILES)

    # Calculate Max Drawdown
    max_drawdown = np.min(
        cumulative_returns_with_zero / np.maximum.
↪accumulate(cumulative_returns_with_zero, axis=1),
        axis=1
    ) - 1
    max_drawdown_percentiles = np.percentile(max_drawdown, PERCENTILES)

    # Create Summary DataFrame
    summary = pd.DataFrame({
        "Portfolio End Balance ($)": end_balances,
        "Annual Compounded Return (%)": annual_compounded_return * 100,
        "Annualized Volatility (%)": annualized_volatility_percentiles * 100,
        "Maximum Drawdown (%)": max_drawdown_percentiles * 100
    }, index=[f"{p}th Percentile" for p in PERCENTILES])

    # Plotting Percentiles
    output_folder = os.path.join("TDIST_MONTE_CARLO_SIMULATIONS", asset_name.
↪replace(' ', '_'))
    os.makedirs(output_folder, exist_ok=True)
    plot_percentiles(cumulative_returns_with_zero, asset_name, output_folder)

    # Style and Export Summary Table
    styled_summary = style_summary_table(summary, f"Monte Carlo Simulation␣
↪Summary for {asset_name}")
    summary_path = os.path.join(output_folder,␣
↪f"{asset_name}_TDIST_summary_table.png")
    export_summary_table(styled_summary, summary_path)

    return cumulative_returns_with_zero, summary, styled_summary

def perform_simulations():
```

```
    """
    Executes Monte Carlo simulations for all assets and displays the combined␣
↪summary.
    """
    # Load Dataset
    try:
        data = pd.read_excel('Opportunity_Set.xlsx')
    except FileNotFoundError:
        print("Error: 'Opportunity_Set.xlsx' file not found.")
        return
    except Exception as e:
        print(f"Error loading dataset: {e}")
        return

    # Identify Assets (Exclude 'Date' Column)
    assets = [col for col in data.columns if col != 'Date']

    if not assets:
        print("No assets found in the dataset.")
        return

    all_assets_summary = []
    cumulative_returns_dict = {}

    # Perform Simulation for Each Asset
    for asset in assets:
        try:
            print(f"Running Monte Carlo Simulation for {asset}")
            asset_returns = data[asset].dropna()
            if asset_returns.empty:
                print(f"Warning: No returns data for {asset}. Skipping.")
                continue
            cumulative_returns, summary, _ =␣
↪run_monte_carlo_simulations(asset_returns, asset)

            # Add Asset Name and Percentile to Summary
            summary = summary.reset_index().rename(columns={"index":␣
↪"Percentile"})
            summary.insert(0, "Asset", asset)
            all_assets_summary.append(summary)
            cumulative_returns_dict[asset] = cumulative_returns
        except Exception as e:
            print(f"An error occurred while processing {asset}: {e}")

    if all_assets_summary:
        # Concatenate All Summaries into a Single DataFrame
        all_assets_summary_df = pd.concat(all_assets_summary, ignore_index=True)
```

```
        # Define Output Folder for Summary
        summary_output_folder = 'TDIST_MONTE_CARLO_SIMULATIONS_SUMMARY'
        os.makedirs(summary_output_folder, exist_ok=True)

        # Style and Export All Assets Summary
        styled_all_summary = style_summary_table(all_assets_summary_df, "Monte␣
↪Carlo Simulation Summary for All Assets")
        all_assets_summary_path = os.path.join(summary_output_folder,␣
↪"TDIST_all_assets_summary_table.png")
        export_summary_table(styled_all_summary, all_assets_summary_path)

        # Display the Styled All Assets Summary in Jupyter Notebook
        display(styled_all_summary)

        # Export Separate Percentile Summary Tables
        for p in PERCENTILES:
            percentile_label = f"{p}th Percentile"
            percentile_df =␣
↪all_assets_summary_df[all_assets_summary_df["Percentile"] ==␣
↪percentile_label]
            if not percentile_df.empty:
                # Drop 'Percentile' Column for Clarity
                percentile_df = percentile_df.drop(columns=["Percentile"])

                # Style and Export
                styled_percentile = style_summary_table(
                    percentile_df,
                    f"TDIST_{p}th Percentile Summary for All Assets"
                )
                percentile_path = os.path.join(summary_output_folder,␣
↪f"TDIST_{p}_percentile_summary_table.png")
                export_summary_table(styled_percentile, percentile_path)

        # Plot Combined Asset Simulations
        plot_combined_assets(cumulative_returns_dict, assets,␣
↪summary_output_folder)
    else:
        print("No simulation summaries to display.")

# Execute the simulations
perform_simulations()
```

```
Running Monte Carlo Simulation for Vanguard LifeStrategy Income Fund (VASIX)
Running Monte Carlo Simulation for Vanguard Total World Stock ETF (VT)
Running Monte Carlo Simulation for PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)
Running Monte Carlo Simulation for AQR Diversified Arbitrage I (ADAIX)
```

```
Running Monte Carlo Simulation for iShares Gold Trust (IAU)
Running Monte Carlo Simulation for Bitcoin Market Price USD (^BTC)
Running Monte Carlo Simulation for AQR Risk-Balanced Commodities Strategy I
(ARCIX)
Running Monte Carlo Simulation for AQR Long-Short Equity I (QLEIX)
Running Monte Carlo Simulation for AQR Style Premia Alternative I (QSPIX)
Running Monte Carlo Simulation for AQR Equity Market Neutral I (QMNIX)
Running Monte Carlo Simulation for AQR Macro Opportunities I (QGMIX)
Running Monte Carlo Simulation for AGF U.S. Market Neutral Anti-Beta (BTAL)
Running Monte Carlo Simulation for AQR Managed Futures Strategy HV I (QMHIX)
Running Monte Carlo Simulation for Invesco DB US Dollar Bullish (UUP)
Running Monte Carlo Simulation for ProShares VIX Mid-Term Futures (VIXM)
```

```
<pandas.io.formats.style.Styler at 0x13111c230>
```

25 Optimized Portfolio Portfolio Weights (Table 11)

26 Optimized Portfolio Risk Contribution (Table 12)

27 Optimized Portfolio Annual Return Table Versus Benchmark (Table 13)

28 Optimized Portfolio Performance Summary Table Versus Benchmark (Table 14)

[200]:
```python
# ALL 3 OPTIMIZATIONS SIDE-BY-SIDE (RP, MV, & MD)

import numpy as np
import pandas as pd
from scipy.optimize import minimize

# === Helper Functions ===

def portfolio_volatility(weights, cov_matrix):
    """Calculate the portfolio volatility."""
    return np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))

def risk_contribution(weights, cov_matrix):
    """Calculate the risk contribution of each asset to the portfolio."""
    total_vol = portfolio_volatility(weights, cov_matrix)
    marginal_contrib = np.dot(cov_matrix, weights)
    return (marginal_contrib * weights) / total_vol

def calculate_cagr(df, column_name):
    """Calculate the Compound Annual Growth Rate (CAGR)."""
    days_diff = (df.index[-1] - df.index[0]).days
```

```
    years = days_diff / 365.25
    cumulative_return = (1 + df[column_name]).prod()
    return (cumulative_return ** (1 / years) - 1) * 100  # Convert to percentage

def calculate_max_drawdown(df, column_name):
    """Calculate the Maximum Drawdown."""
    cumulative_return = (1 + df[column_name]).cumprod()
    rolling_max = cumulative_return.cummax()
    drawdown = (cumulative_return - rolling_max) / rolling_max
    return drawdown.min() * 100  # Convert to percentage

def calculate_annual_returns(df, column_name):
    """Calculate annual returns for the portfolio."""
    annual_returns = df[column_name].resample('YE').apply(lambda x: (1 + x).
 ↪prod() - 1)
    return (annual_returns * 100).round(2)  # Convert to percentage and round

# === Portfolio Optimizations ===

def optimize_risk_parity(cov_matrix):
    def risk_parity_objective(weights, cov_matrix):
        contribs = risk_contribution(weights, cov_matrix)
        return np.std(contribs)

    constraints = {'type': 'eq', 'fun': lambda w: np.sum(w) - 1}
    bounds = [(0, 1) for _ in range(len(cov_matrix))]
    init_guess = np.ones(len(cov_matrix)) / len(cov_matrix)

    result = minimize(risk_parity_objective, init_guess, args=(cov_matrix,),␣
 ↪method='SLSQP', bounds=bounds, constraints=constraints)
    return result.x

def optimize_min_variance(cov_matrix):
    def min_variance_objective(weights, cov_matrix):
        return portfolio_volatility(weights, cov_matrix)

    constraints = {'type': 'eq', 'fun': lambda w: np.sum(w) - 1}
    bounds = [(0, 1) for _ in range(len(cov_matrix))]
    init_guess = np.ones(len(cov_matrix)) / len(cov_matrix)

    result = minimize(min_variance_objective, init_guess, args=(cov_matrix,),␣
 ↪method='SLSQP', bounds=bounds, constraints=constraints)
    return result.x

def optimize_max_diversification(cov_matrix, asset_std):
    def diversification_ratio(weights, asset_std, cov_matrix):
        weighted_volatility_sum = np.dot(weights, asset_std)
```

```
        portfolio_vol = portfolio_volatility(weights, cov_matrix)
        return weighted_volatility_sum / portfolio_vol

    def max_diversification_objective(weights, asset_std, cov_matrix):
        return -diversification_ratio(weights, asset_std, cov_matrix)

    constraints = {'type': 'eq', 'fun': lambda w: np.sum(w) - 1}
    bounds = [(0, 1) for _ in range(len(cov_matrix))]
    init_guess = np.ones(len(cov_matrix)) / len(cov_matrix)

    result = minimize(max_diversification_objective, init_guess,␣
 ↪args=(asset_std, cov_matrix), method='SLSQP', bounds=bounds,␣
 ↪constraints=constraints)
    return result.x

# === Load Data ===

# Define the file path
file_path = 'Opportunity_Set.xlsx'

# Read the Excel file, parse 'Date' as datetime, and set it as the index
data = pd.read_excel(file_path, parse_dates=['Date'], index_col='Date')

# Exclude the benchmark and keep only asset columns
asset_columns = [col for col in data.columns if col != 'Vanguard LifeStrategy␣
 ↪Income Fund (VASIX)']
assets = data[asset_columns].apply(pd.to_numeric, errors='coerce').dropna()

# Calculate the covariance matrix and asset volatilities (std)
cov_matrix = assets.cov()
asset_std = assets.std()

# === Perform Optimizations for all Portfolios ===

weights_risk_parity = optimize_risk_parity(cov_matrix)
weights_min_variance = optimize_min_variance(cov_matrix)
weights_max_diversification = optimize_max_diversification(cov_matrix,␣
 ↪asset_std)

# === Portfolio Returns ===

# Calculate returns using the optimized weights
returns_risk_parity = assets.dot(weights_risk_parity)
returns_min_variance = assets.dot(weights_min_variance)
returns_max_diversification = assets.dot(weights_max_diversification)

# Combine portfolio returns into a DataFrame and include VASIX benchmark returns
```

```
performance_df = pd.DataFrame({
    'Risk Parity': returns_risk_parity,
    'Minimum Variance': returns_min_variance,
    'Maximum Diversification': returns_max_diversification,
    'VASIX (Benchmark)': data['Vanguard LifeStrategy Income Fund (VASIX)']
})

# === Build Tables for Weights and Risk Contributions ===

def build_weight_table():
    """Create a table for weights only."""
    table = pd.DataFrame({
        'Asset': asset_columns,
        'Risk Parity': weights_risk_parity * 100,
        'Min Variance': weights_min_variance * 100,
        'Max Diversification': weights_max_diversification * 100
    }).round(2)
    return table

def build_risk_contrib_table():
    """Create a table for risk contributions only."""
    risk_contrib_rp = risk_contribution(weights_risk_parity, cov_matrix) /␣
 ↪portfolio_volatility(weights_risk_parity, cov_matrix) * 100
    risk_contrib_mv = risk_contribution(weights_min_variance, cov_matrix) /␣
 ↪portfolio_volatility(weights_min_variance, cov_matrix) * 100
    risk_contrib_md = risk_contribution(weights_max_diversification,␣
 ↪cov_matrix) / portfolio_volatility(weights_max_diversification, cov_matrix)␣
 ↪* 100

    table = pd.DataFrame({
        'Asset': asset_columns,
        'Risk Parity': risk_contrib_rp,
        'Min Variance': risk_contrib_mv,
        'Max Diversification': risk_contrib_md
    }).round(2)
    return table

# === Annual Returns Table ===

def build_annual_returns_table():
    """Create a table for annual returns."""
    annual_returns_rp = calculate_annual_returns(performance_df, 'Risk Parity')
    annual_returns_mv = calculate_annual_returns(performance_df, 'Minimum␣
 ↪Variance')
    annual_returns_md = calculate_annual_returns(performance_df, 'Maximum␣
 ↪Diversification')
```

```
    annual_returns_vasix = calculate_annual_returns(performance_df, 'VASIX␣
↪(Benchmark)')

    table = pd.DataFrame({
        'Risk Parity (%)': annual_returns_rp,
        'Minimum Variance (%)': annual_returns_mv,
        'Maximum Diversification (%)': annual_returns_md,
        'VASIX (Benchmark) (%)': annual_returns_vasix
    })
    return table

# === Performance Metrics Table ===

def build_performance_metrics(df, name):
    """Calculate CAGR, Standard Deviation, and Maximum Drawdown."""
    cagr = calculate_cagr(df, name)
    std_dev = df[name].std() * np.sqrt(12) * 100  # Annualized standard␣
↪deviation
    mdd = calculate_max_drawdown(df, name)

    return pd.Series({'CAGR (%)': cagr, 'Standard Deviation (%)': std_dev,␣
↪'Maximum Drawdown (%)': mdd}).round(2)

def build_performance_metrics_table():
    """Create a table for performance metrics."""
    performance_metrics = pd.DataFrame({
        'Risk Parity': build_performance_metrics(performance_df, 'Risk Parity'),
        'Minimum Variance': build_performance_metrics(performance_df, 'Minimum␣
↪Variance'),
        'Maximum Diversification': build_performance_metrics(performance_df,␣
↪'Maximum Diversification'),
        'VASIX (Benchmark)': build_performance_metrics(performance_df, 'VASIX␣
↪(Benchmark)')
    })
    return performance_metrics

# === Generate Tables ===

# Generate the tables
weights_table = build_weight_table()
risk_contrib_table = build_risk_contrib_table()
annual_returns_table = build_annual_returns_table()
performance_metrics_table = build_performance_metrics_table()

# Display the tables
print("Portfolio Weights Table (%):")
```

```
print(weights_table.to_string(index=False))

print("\nRisk Contribution Table (%):")
print(risk_contrib_table.to_string(index=False))

print("\nAnnual Return Table (%):")
print(annual_returns_table.to_string())

print("\nPerformance Metrics Table:")
print(performance_metrics_table.round(2).to_string())
```

```
Portfolio Weights Table (%):
                                        Asset  Risk Parity  Min Variance  Max
Diversification
             Vanguard Total World Stock ETF (VT)        11.62         15.84
23.15
    PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)         4.75          2.01
3.38
             AQR Diversified Arbitrage I (ADAIX)        15.04         22.46
8.49
                         iShares Gold Trust (IAU)         4.05          3.29
0.00
                 Bitcoin Market Price USD (^BTC)         0.92          0.00
0.46
AQR Risk-Balanced Commodities Strategy I (ARCIX)         4.27          0.31
4.08
                 AQR Long-Short Equity I (QLEIX)         4.58          0.00
0.00
          AQR Style Premia Alternative I (QSPIX)         3.82          0.00
0.00
             AQR Equity Market Neutral I (QMNIX)         5.59          9.09
9.16
               AQR Macro Opportunities I (QGMIX)         8.89          7.80
8.07
        AGF U.S. Market Neutral Anti-Beta (BTAL)         6.13          6.69
8.73
       AQR Managed Futures Strategy HV I (QMHIX)         2.68          0.00
0.00
              Invesco DB US Dollar Bullish (UUP)        22.30         27.87
26.54
           ProShares VIX Mid-Term Futures (VIXM)         5.36          4.64
7.92

Risk Contribution Table (%):
                                        Asset  Risk Parity  Min Variance  Max
Diversification
             Vanguard Total World Stock ETF (VT)         7.19         16.90
27.12
```

```
    PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)         7.11          1.94
5.54
             AQR Diversified Arbitrage I (ADAIX)         7.09         22.05
3.87
                         iShares Gold Trust (IAU)         7.14          3.30
0.00
                  Bitcoin Market Price USD (^BTC)         7.10          0.00
2.74
AQR Risk-Balanced Commodities Strategy I (ARCIX)         7.18          0.29
5.16
                  AQR Long-Short Equity I (QLEIX)         7.15          0.00
0.00
           AQR Style Premia Alternative I (QSPIX)         7.16          0.00
0.00
              AQR Equity Market Neutral I (QMNIX)         7.14          9.25
7.09
                AQR Macro Opportunities I (QGMIX)         7.16          7.90
4.74
         AGF U.S. Market Neutral Anti-Beta (BTAL)         7.11          6.49
10.08
        AQR Managed Futures Strategy HV I (QMHIX)         7.17          0.00
0.00
               Invesco DB US Dollar Bullish (UUP)         7.16         27.48
13.68
            ProShares VIX Mid-Term Futures (VIXM)         7.14          4.40
19.98

Annual Return Table (%):
             Risk Parity (%)  Minimum Variance (%)  Maximum Diversification (%)
VASIX (Benchmark) (%)
Date
2014-12-31             2.11                  1.77                         1.77
0.89
2015-12-31             1.36                  1.39                         1.29
0.23
2016-12-31             4.17                  3.82                         3.35
4.60
2017-12-31             3.51                 -0.03                         0.39
6.97
2018-12-31            -0.34                  2.60                         1.60
-1.06
2019-12-31             7.19                  6.64                         7.48
12.05
2020-12-31             9.11                  8.56                         9.61
9.14
2021-12-31             7.99                  5.97                         7.47
1.92
2022-12-31             5.71                  4.01                         3.29
```

```
-13.93
2023-12-31             4.62                  4.07                          3.03
9.48
2024-12-31             7.27                  6.92                          6.94
5.21

Performance Metrics Table:
                        Risk Parity  Minimum Variance  Maximum Diversification
VASIX (Benchmark)
CAGR (%)                       5.38              4.66                        4.71
3.38
Standard Deviation (%)         2.93              2.35                        2.61
5.63
Maximum Drawdown (%)          -2.87             -2.21                       -2.77
-16.72
```

29 Optimized Portfolio Performance Regression Against Benchmark (Table 15)

[40]:
```python
import numpy as np
import pandas as pd
from scipy.optimize import minimize
import statsmodels.api as sm
import os

# Optional: For exporting styled tables as images
try:
    import dataframe_image as dfi
except ImportError:
    dfi = None  # Handle gracefully if not installed

# === Set Global Pandas Display Options ===
pd.options.display.float_format = '{:.2f}'.format

# === Helper Functions (No Changes) ===
def portfolio_volatility(weights, cov_matrix):
    return np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))

def risk_contribution(weights, cov_matrix):
    total_vol = portfolio_volatility(weights, cov_matrix)
    marginal_contrib = np.dot(cov_matrix, weights)
    return (marginal_contrib * weights) / total_vol

def calculate_cagr(df, column_name):
    days_diff = (df.index[-1] - df.index[0]).days
    years = days_diff / 365.25
```

```
    cumulative_return = (1 + df[column_name]).prod()
    return round((cumulative_return ** (1 / years) - 1) * 100, 2)

def calculate_max_drawdown(df, column_name):
    cumulative_return = (1 + df[column_name]).cumprod()
    rolling_max = cumulative_return.cummax()
    drawdown = (cumulative_return - rolling_max) / rolling_max
    return round(drawdown.min() * 100, 2)

def calculate_annual_returns(df, column_name):
    annual_returns = df[column_name].resample('YE').apply(lambda x: (1 + x).
 ↪prod() - 1)
    return (annual_returns * 100).round(2)

# === Portfolio Optimizations (No Changes) ===
def optimize_risk_parity(cov_matrix):
    def risk_parity_objective(weights, cov_matrix):
        contribs = risk_contribution(weights, cov_matrix)
        return np.std(contribs)
    constraints = {'type': 'eq', 'fun': lambda w: np.sum(w) - 1}
    bounds = [(0, 1) for _ in range(len(cov_matrix))]
    init_guess = np.ones(len(cov_matrix)) / len(cov_matrix)
    result = minimize(risk_parity_objective, init_guess, args=(cov_matrix,),␣
 ↪method='SLSQP', bounds=bounds, constraints=constraints)
    return result.x

def optimize_min_variance(cov_matrix):
    def min_variance_objective(weights, cov_matrix):
        return portfolio_volatility(weights, cov_matrix)
    constraints = {'type': 'eq', 'fun': lambda w: np.sum(w) - 1}
    bounds = [(0, 1) for _ in range(len(cov_matrix))]
    init_guess = np.ones(len(cov_matrix)) / len(cov_matrix)
    result = minimize(min_variance_objective, init_guess, args=(cov_matrix,),␣
 ↪method='SLSQP', bounds=bounds, constraints=constraints)
    return result.x

def optimize_max_diversification(cov_matrix, asset_std):
    def diversification_ratio(weights, asset_std, cov_matrix):
        weighted_volatility_sum = np.dot(weights, asset_std)
        portfolio_vol = portfolio_volatility(weights, cov_matrix)
        return weighted_volatility_sum / portfolio_vol
    def max_diversification_objective(weights, asset_std, cov_matrix):
        return -diversification_ratio(weights, asset_std, cov_matrix)
    constraints = {'type': 'eq', 'fun': lambda w: np.sum(w) - 1}
    bounds = [(0, 1) for _ in range(len(cov_matrix))]
    init_guess = np.ones(len(cov_matrix)) / len(cov_matrix)
```

```
    result = minimize(max_diversification_objective, init_guess,␣
 ↪args=(asset_std, cov_matrix), method='SLSQP', bounds=bounds,␣
 ↪constraints=constraints)
    return result.x

# === Load Data ===
file_path = 'Opportunity_Set.xlsx'
data = pd.read_excel(file_path, parse_dates=['Date'], index_col='Date')
asset_columns = [col for col in data.columns if col != 'Vanguard LifeStrategy␣
 ↪Income Fund (VASIX)']
assets = data[asset_columns].apply(pd.to_numeric, errors='coerce').dropna()

# Calculate the covariance matrix and asset volatilities (std)
cov_matrix = assets.cov()
asset_std = assets.std()

# === Perform Optimizations for all Portfolios ===
weights_risk_parity = optimize_risk_parity(cov_matrix)
weights_min_variance = optimize_min_variance(cov_matrix)
weights_max_diversification = optimize_max_diversification(cov_matrix,␣
 ↪asset_std)

# === Portfolio Returns ===
returns_risk_parity = assets.dot(weights_risk_parity)
returns_min_variance = assets.dot(weights_min_variance)
returns_max_diversification = assets.dot(weights_max_diversification)

performance_df = pd.DataFrame({
    'Risk Parity': returns_risk_parity,
    'Minimum Variance': returns_min_variance,
    'Maximum Diversification': returns_max_diversification,
    'VASIX (Benchmark)': data['Vanguard LifeStrategy Income Fund (VASIX)']
})

# === Regression Analysis for Portfolios ===
portfolios = ['Risk Parity', 'Minimum Variance', 'Maximum Diversification']
benchmark = performance_df['VASIX (Benchmark)'].dropna()

regression_results = []

for portfolio in portfolios:
    portfolio_returns = performance_df[portfolio].dropna()
    combined_data = pd.concat([benchmark, portfolio_returns], axis=1).dropna()
    X = combined_data['VASIX (Benchmark)']
    Y = combined_data[portfolio]
    X = sm.add_constant(X)
    model = sm.OLS(Y, X).fit()
```

```
    intercept = round(model.params['const'] * 12 * 100, 2)
    intercept_tstat = round(model.tvalues['const'], 2)
    intercept_pvalue = round(model.pvalues['const'], 2)
    beta = round(model.params['VASIX (Benchmark)'], 2)
    beta_tstat = round(model.tvalues['VASIX (Benchmark)'], 2)
    beta_pvalue = round(model.pvalues['VASIX (Benchmark)'], 2)

    arithmetic_return = round(portfolio_returns.mean() * 12 * 100, 2)

    regression_results.append({
        "Portfolio": portfolio,
        "R2": round(model.rsquared, 2),
        "Annualized Arithmetic Return (%)": arithmetic_return,
        "Intercept (Annualized)": intercept,
        "Intercept t-stat": intercept_tstat,
        "Intercept p-value": intercept_pvalue,
        "Beta": beta,
        "Beta t-stat": beta_tstat,
        "Beta p-value": beta_pvalue
    })

regression_df = pd.DataFrame(regression_results)

# === Style and Display Regression Table ===
def highlight_significant(row):
    styles = []
    if row['Intercept p-value'] < 0.05:
        styles.extend(['font-weight: bold'] * 3)
    else:
        styles.extend([''] * 3)
    if row['Beta p-value'] < 0.05:
        styles.extend(['font-weight: bold'] * 3)
    else:
        styles.extend([''] * 3)
    return styles

def style_intercept(val):
    return 'background-color: #e6f2ff; border-right: 2px solid #000'

def style_beta(val):
    return 'background-color: #e6ffe6; border-right: 2px solid #000'

def style_r2(val):
    return 'background-color: #ffffe6'

styled_regression_table = (
```

```
    regression_df.style
    .format({
        "Intercept (Annualized)": "{:.2f}",
        "Intercept t-stat": "{:.2f}",
        "Intercept p-value": "{:.2f}",
        "Beta": "{:.2f}",
        "Beta t-stat": "{:.2f}",
        "Beta p-value": "{:.2f}",
        "R2": "{:.2f}",
        "Annualized Arithmetic Return (%)": "{:.2f}"
    })
    .apply(highlight_significant, axis=1, subset=[
        'Intercept (Annualized)', 'Intercept t-stat', 'Intercept p-value',␣
 ↪'Beta', 'Beta t-stat', 'Beta p-value'
    ])
    .apply(lambda x: [style_intercept(val) for val in x], subset=[
        'Intercept (Annualized)', 'Intercept t-stat', 'Intercept p-value'
    ])
    .apply(lambda x: [style_beta(val) for val in x], subset=[
        'Beta', 'Beta t-stat', 'Beta p-value'
    ])
    .apply(lambda x: [style_r2(val) for val in x], subset=['R2'])
    .set_properties(
        subset=['R2', 'Annualized Arithmetic Return (%)', 'Intercept␣
 ↪(Annualized)', 'Intercept t-stat',
                'Intercept p-value', 'Beta', 'Beta t-stat', 'Beta p-value'],
        **{'text-align': 'center'}
    )
    .set_table_styles([
        {'selector': 'th', 'props': [('text-align', 'center'), ('white-space',␣
 ↪'pre-wrap')]},
        {'selector': 'td, th', 'props': [('border', '1px solid black')]},
        {'selector': 'td.Portfolio', 'props': [('text-align', 'left'),␣
 ↪('white-space', 'nowrap')]}
    ])
    .set_caption("Regression Results Against Benchmark (VASIX)")
)

# Display the styled regression table (the only output)
try:
    from IPython.display import display
    display(styled_regression_table)
except ImportError:
    pass  # Not in an environment that supports rich display

# Optionally, save the styled table as an image
if dfi:
```

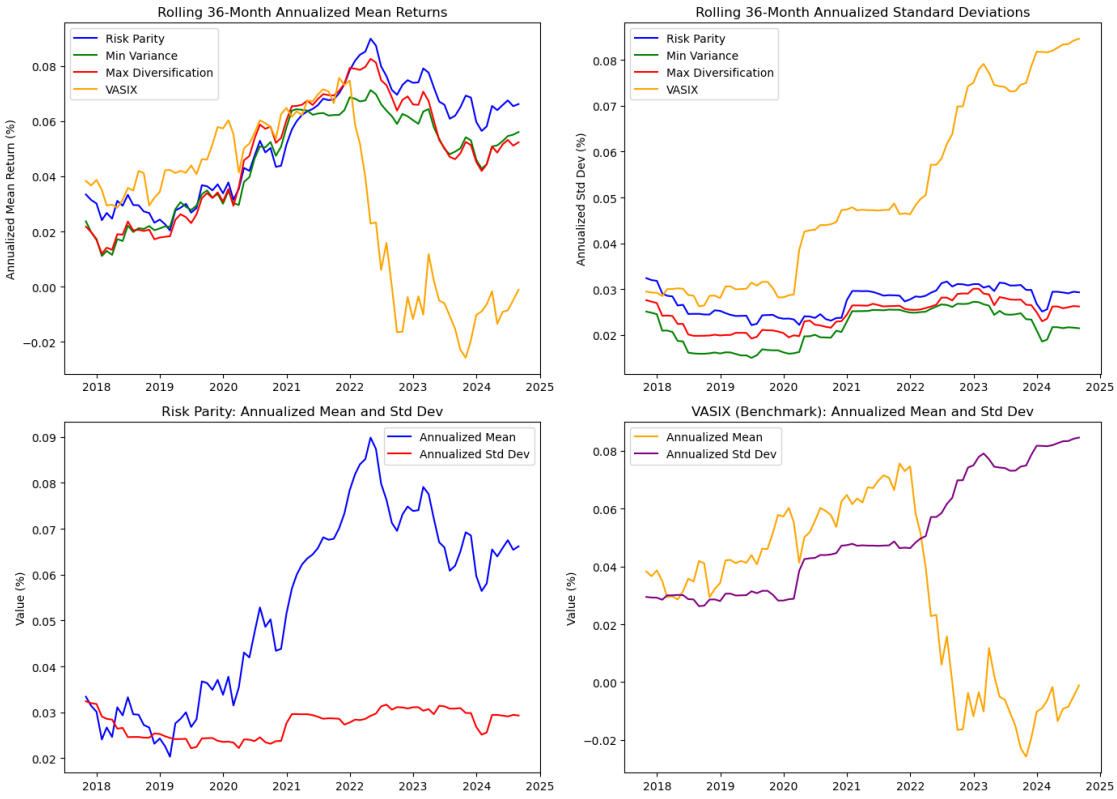
```
    regression_output_folder = 'REGRESSION_RESULTS'
    os.makedirs(regression_output_folder, exist_ok=True)
    regression_output_file_path = os.path.join(regression_output_folder,␣
 ↪"regression_results_table.png")
    try:
        dfi.export(styled_regression_table, regression_output_file_path)
    except Exception as e:
        pass  # Handle any issues with saving the image
```

```
<pandas.io.formats.style.Styler at 0x3438c0e30>
```

30 Optimized Portfolio Rolling 36-month Mean and Standard Deviations (Figure 10)

[230]:
```
import matplotlib.pyplot as plt
import os

# === Calculate Rolling 36-Month Mean and Standard Deviation, Annualized ===

# Define a 36-month window
rolling_window = 36

# Calculate rolling mean and standard deviation for each portfolio and benchmark
rolling_stats = performance_df.rolling(window=rolling_window).agg(['mean',␣
 ↪'std'])

# Extract the rolling means and standard deviations for each column, then␣
 ↪annualize
rolling_means_annualized = rolling_stats.xs('mean', level=1, axis=1) * 12  #␣
 ↪Annualize monthly mean returns
rolling_stds_annualized = rolling_stats.xs('std', level=1, axis=1) * np.
 ↪sqrt(12)  # Annualize monthly std devs

# === Plot Rolling Mean and Standard Deviation in a 2x2 Layout ===

fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# Plot rolling mean (annualized)
axes[0, 0].plot(rolling_means_annualized['Risk Parity'], label='Risk Parity',␣
 ↪color='blue')
axes[0, 0].plot(rolling_means_annualized['Minimum Variance'], label='Min␣
 ↪Variance', color='green')
axes[0, 0].plot(rolling_means_annualized['Maximum Diversification'], label='Max␣
 ↪Diversification', color='red')
axes[0, 0].plot(rolling_means_annualized['VASIX (Benchmark)'], label='VASIX',␣
 ↪color='orange')
```

```
axes[0, 0].set_title('Rolling 36-Month Annualized Mean Returns')
axes[0, 0].set_ylabel('Annualized Mean Return (%)')
axes[0, 0].legend()

# Plot rolling standard deviation (annualized)
axes[0, 1].plot(rolling_stds_annualized['Risk Parity'], label='Risk Parity',␣
 ↪color='blue')
axes[0, 1].plot(rolling_stds_annualized['Minimum Variance'], label='Min␣
 ↪Variance', color='green')
axes[0, 1].plot(rolling_stds_annualized['Maximum Diversification'], label='Max␣
 ↪Diversification', color='red')
axes[0, 1].plot(rolling_stds_annualized['VASIX (Benchmark)'], label='VASIX',␣
 ↪color='orange')
axes[0, 1].set_title('Rolling 36-Month Annualized Standard Deviations')
axes[0, 1].set_ylabel('Annualized Std Dev (%)')
axes[0, 1].legend()

# Combined plot for Risk Parity (annualized mean and std dev)
axes[1, 0].plot(rolling_means_annualized['Risk Parity'], label='Annualized␣
 ↪Mean', color='blue')
axes[1, 0].plot(rolling_stds_annualized['Risk Parity'], label='Annualized Std␣
 ↪Dev', color='red')
axes[1, 0].set_title('Risk Parity: Annualized Mean and Std Dev')
axes[1, 0].set_ylabel('Value (%)')
axes[1, 0].legend()

# Combined plot for Benchmark (VASIX, annualized mean and std dev)
axes[1, 1].plot(rolling_means_annualized['VASIX (Benchmark)'],␣
 ↪label='Annualized Mean', color='orange')
axes[1, 1].plot(rolling_stds_annualized['VASIX (Benchmark)'], label='Annualized␣
 ↪Std Dev', color='purple')
axes[1, 1].set_title('VASIX (Benchmark): Annualized Mean and Std Dev')
axes[1, 1].set_ylabel('Value (%)')
axes[1, 1].legend()

# Adjust layout
plt.tight_layout()

# Save the plot as a PNG file in the active directory
output_filename = 'rolling_36_month_mean_std_plot.png'
plt.savefig(output_filename)

# Show the plot
plt.show()

print(f"Plot saved as {output_filename}")
```



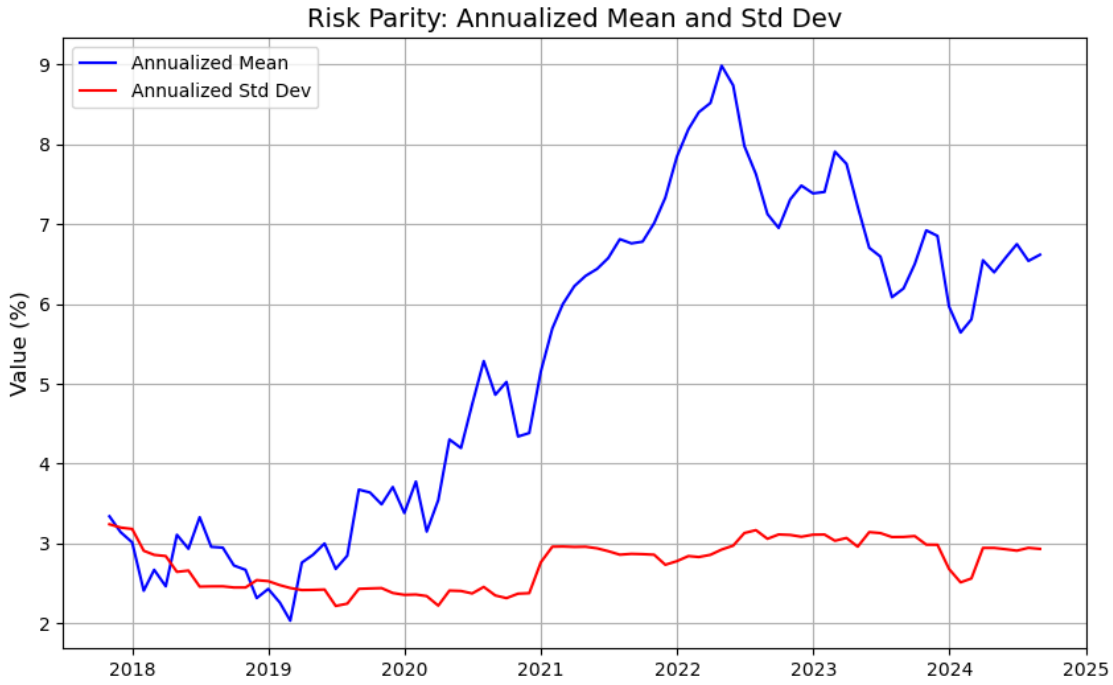

```
Plot saved as rolling_36_month_mean_std_plot.png
```

31 Individual Optimized Portfolio Rolling 36-month Mean and Standard Deviations (Not Shown in Report)

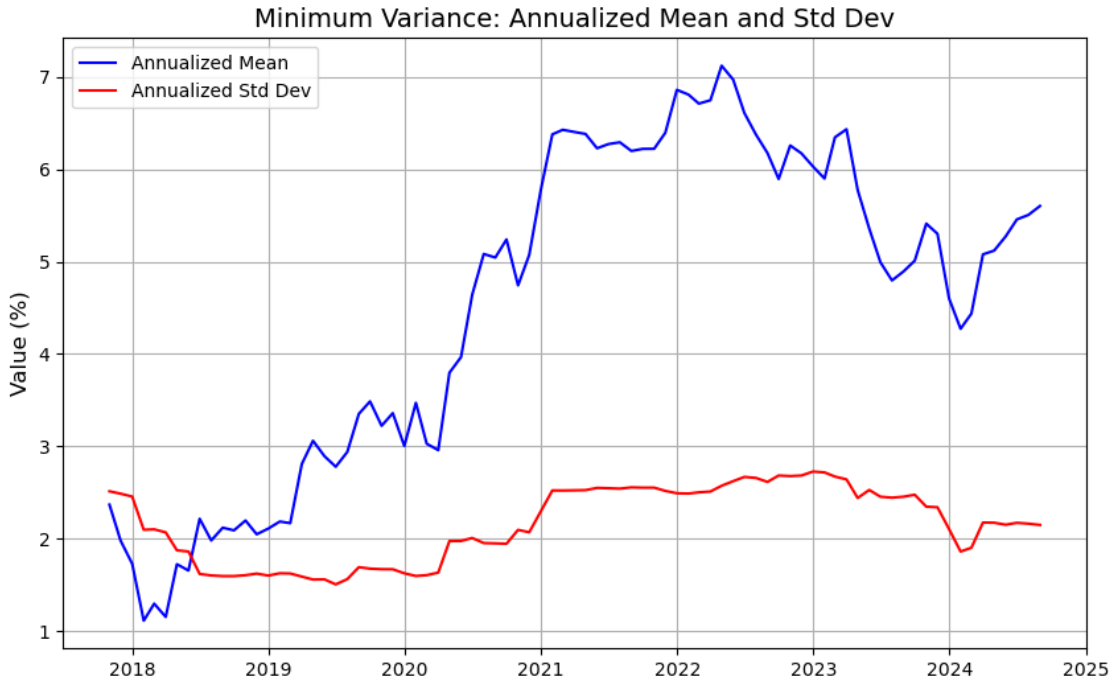
[50]:
```
import matplotlib.pyplot as plt
import os

# Create a folder for saving the outputted PNG files if it doesn't exist
output_folder = 'Rolling_Stats_Plots'
os.makedirs(output_folder, exist_ok=True)

# === Calculate Rolling 36-Month Mean and Standard Deviation, Annualized ===

# Define a 36-month window
rolling_window = 36

# Calculate rolling mean and standard deviation for each portfolio and benchmark
rolling_stats = performance_df.rolling(window=rolling_window).agg(['mean',␣
↪'std'])
```

```
# Extract the rolling means and standard deviations for each column, then␣
 ↪annualize and multiply by 100 for percentage
rolling_means_annualized = rolling_stats.xs('mean', level=1, axis=1) * 12 * 100␣
 ↪ # Annualize and express as percentage
rolling_stds_annualized = rolling_stats.xs('std', level=1, axis=1) * np.
 ↪sqrt(12) * 100  # Annualize and express as percentage

# === Individual Plots for Each Portfolio ===

portfolios = ['Risk Parity', 'Minimum Variance', 'Maximum Diversification',␣
 ↪'VASIX (Benchmark)']

for portfolio in portfolios:
    fig, ax = plt.subplots(figsize=(10, 6))
    ax.plot(rolling_means_annualized[portfolio], label='Annualized Mean',␣
 ↪color='blue')
    ax.plot(rolling_stds_annualized[portfolio], label='Annualized Std Dev',␣
 ↪color='red')
    ax.set_title(f'{portfolio}: Annualized Mean and Std Dev')
    ax.set_ylabel('Value (%)')
    ax.legend(loc='upper left')
    ax.grid(True)

    # Save individual plots
    output_filename = os.path.join(output_folder, f'{portfolio.lower().
 ↪replace(" ", "_")}_rolling_stats.png')
    plt.savefig(output_filename)

    # Display the plot in Jupyter Notebook
    plt.show()  # Added to display the plot

    plt.close()  # Close the plot after saving

    print(f"Individual plot saved as {output_filename}")

# === Combined Plot for All Portfolios (Mean and Std Dev) ===

fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Set consistent plot styles
plt.rcParams.update({
    'axes.titlesize': 14,
    'axes.labelsize': 12,
    'legend.fontsize': 10,
    'xtick.labelsize': 10,
    'ytick.labelsize': 10
```

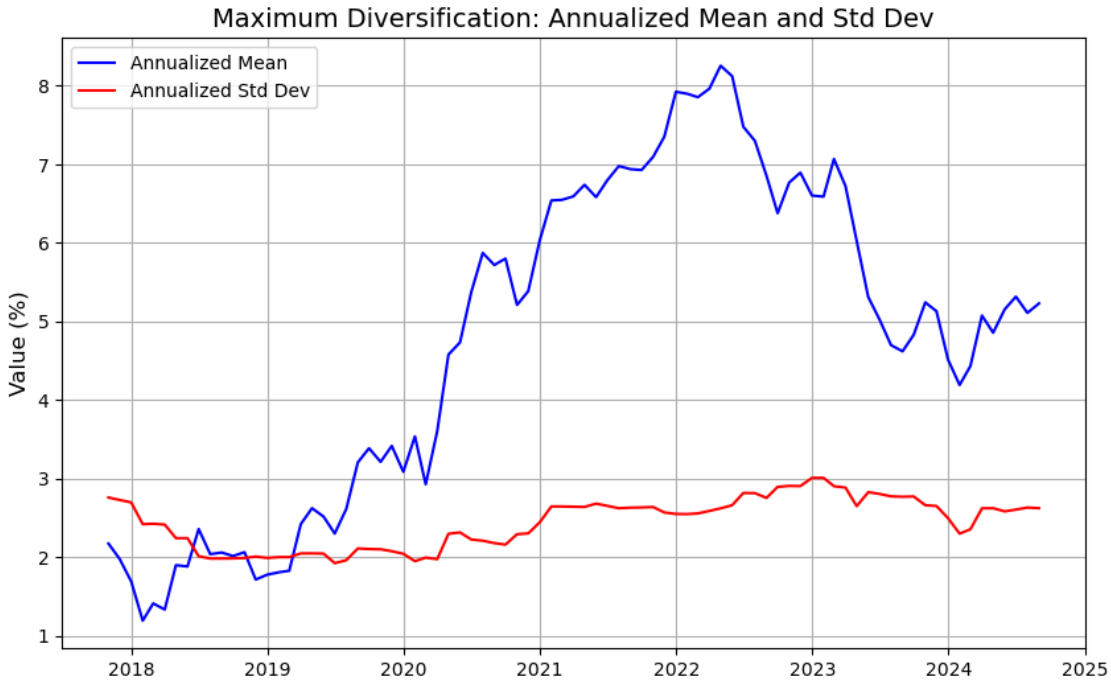
```
})

# Combined plot for rolling mean (annualized, percentage)
axes[0].plot(rolling_means_annualized['Risk Parity'], label='Risk Parity',␣
 ↪color='blue')
axes[0].plot(rolling_means_annualized['Minimum Variance'], label='Min␣
 ↪Variance', color='green')
axes[0].plot(rolling_means_annualized['Maximum Diversification'], label='Max␣
 ↪Diversification', color='red')
axes[0].plot(rolling_means_annualized['VASIX (Benchmark)'], label='VASIX',␣
 ↪color='orange')
axes[0].set_title('Rolling 36-Month Annualized Mean Returns')
axes[0].set_ylabel('Annualized Mean Return (%)')
axes[0].legend(loc='upper left')
axes[0].grid(True)

# Combined plot for rolling standard deviation (annualized, percentage)
axes[1].plot(rolling_stds_annualized['Risk Parity'], label='Risk Parity',␣
 ↪color='blue')
axes[1].plot(rolling_stds_annualized['Minimum Variance'], label='Min Variance',␣
 ↪color='green')
axes[1].plot(rolling_stds_annualized['Maximum Diversification'], label='Max␣
 ↪Diversification', color='red')
axes[1].plot(rolling_stds_annualized['VASIX (Benchmark)'], label='VASIX',␣
 ↪color='orange')
axes[1].set_title('Rolling 36-Month Annualized Standard Deviations')
axes[1].set_ylabel('Annualized Std Dev (%)')
axes[1].legend(loc='upper left')
axes[1].grid(True)

# Adjust layout
plt.tight_layout()

# Save the combined plot as a PNG file
combined_plot_filename = os.path.join(output_folder, 'combined_rolling_mean_std.
 ↪png')
plt.savefig(combined_plot_filename)

# Display the combined plot in Jupyter Notebook
plt.show()  # Added to display the combined plot

plt.close()  # Close the plot after saving

print(f"Combined plot saved as {combined_plot_filename}")
```



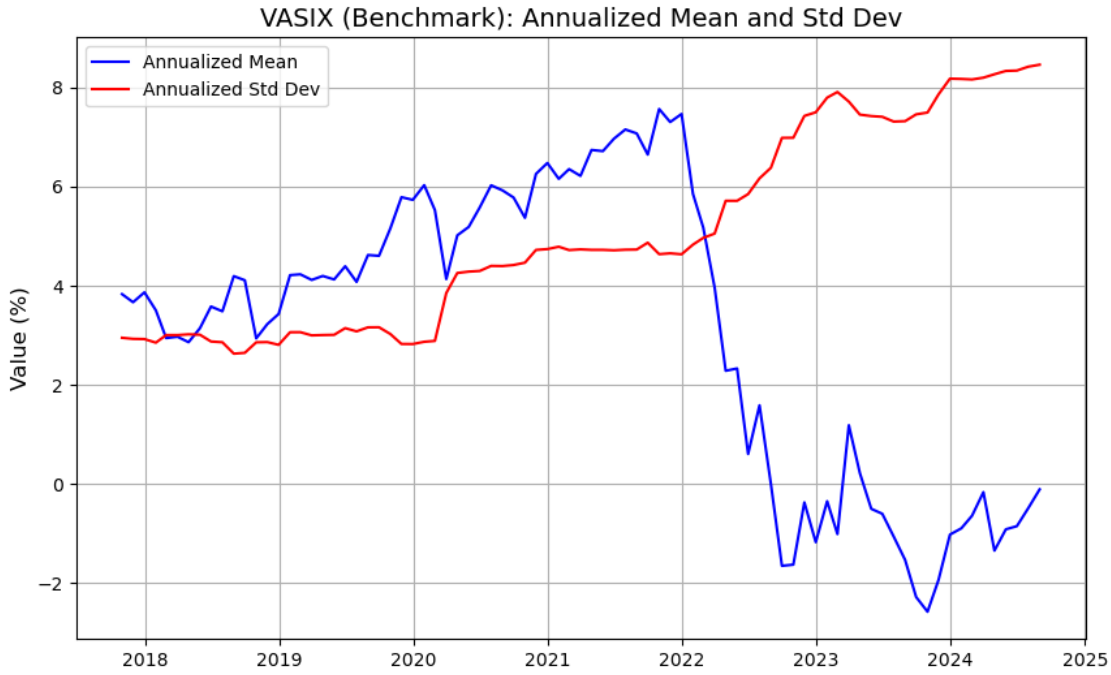

```
Individual plot saved as Rolling_Stats_Plots/risk_parity_rolling_stats.png
```



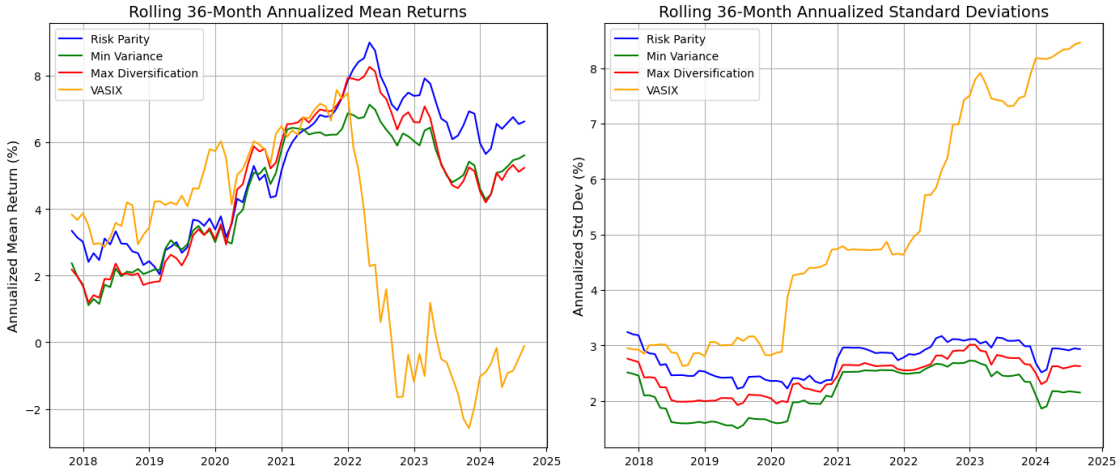

```
Individual plot saved as Rolling_Stats_Plots/minimum_variance_rolling_stats.png
```




```
Individual plot saved as
Rolling_Stats_Plots/maximum_diversification_rolling_stats.png
```



```
Individual plot saved as Rolling_Stats_Plots/vasix_(benchmark)_rolling_stats.png
```




```
Combined plot saved as Rolling_Stats_Plots/combined_rolling_mean_std.png
```

32 Optimized Portfolio Summary Statistics (Table 16)

[41]:
```
import pandas as pd
import numpy as np
import dataframe_image as dfi

# performance_df contains 'Risk Parity', 'Minimum Variance', 'Maximum␣
 ↪Diversification', and 'VASIX (Benchmark)'

# Step 1: Calculate summary statistics using describe()
summary_stats = performance_df.describe().T  # Transpose for readability

# Rename columns for better clarity
summary_stats.columns = ['Count', 'Mean', 'Std Dev', 'Min', '25%', 'Median',␣
 ↪'75%', 'Max']

# Step 2: Add Skewness and Kurtosis
summary_stats['Skewness'] = performance_df.skew()
summary_stats['Kurtosis'] = performance_df.kurtosis()

# Step 3: Compute the correlation matrix and extract the correlation with VASIX
correlation_with_vasix = performance_df.corr()['VASIX (Benchmark)']  #␣
 ↪Automatically includes VASIX self-correlation

# Insert the correlation as a new column in summary_stats
summary_stats.insert(1, 'Correlation with VASIX', correlation_with_vasix)
```

```
# Step 4: Convert Count to integer and percentage-relevant columns to␣
 ↪percentage form
summary_stats['Count'] = summary_stats['Count'].astype(int)

columns_to_convert = ['Mean', 'Std Dev', 'Min', '25%', 'Median', '75%', 'Max']
for col in columns_to_convert:
    summary_stats[col] = summary_stats[col] * 100  # Convert to percentage form

# Step 5: Format and style the table
styled_table = summary_stats.style \
    .format("{:.2f}", subset=pd.IndexSlice[:, columns_to_convert]) \
    .format("{:.0f}", subset=['Count']) \
    .format("{:.2f}", subset=['Correlation with VASIX', 'Skewness',␣
 ↪'Kurtosis']) \
    .set_caption("Summary Statistics for Optimized Portfolios and VASIX") \
    .set_table_styles([
        {'selector': 'caption', 'props': 'caption-side: top; font-size: 14px;␣
 ↪font-weight: bold; color: #6B6B6B;'},
        {'selector': 'thead th', 'props': [('background-color', '#f5f5f5'),␣
 ↪('color', 'black'), ('font-weight', 'bold')]}
    ]) \
    .background_gradient(cmap='Oranges', subset=['Correlation with VASIX'],␣
 ↪axis=0, low=0.2, high=0.8)

# Step 6: Export the styled table as an image (PNG format)
dfi.export(styled_table, 'optimized_portfolios_summary_with_correlation_fixed.
 ↪png')

# Optional: Display the formatted table in Jupyter
styled_table
```

[41]: <pandas.io.formats.style.Styler at 0x13e683cb0>

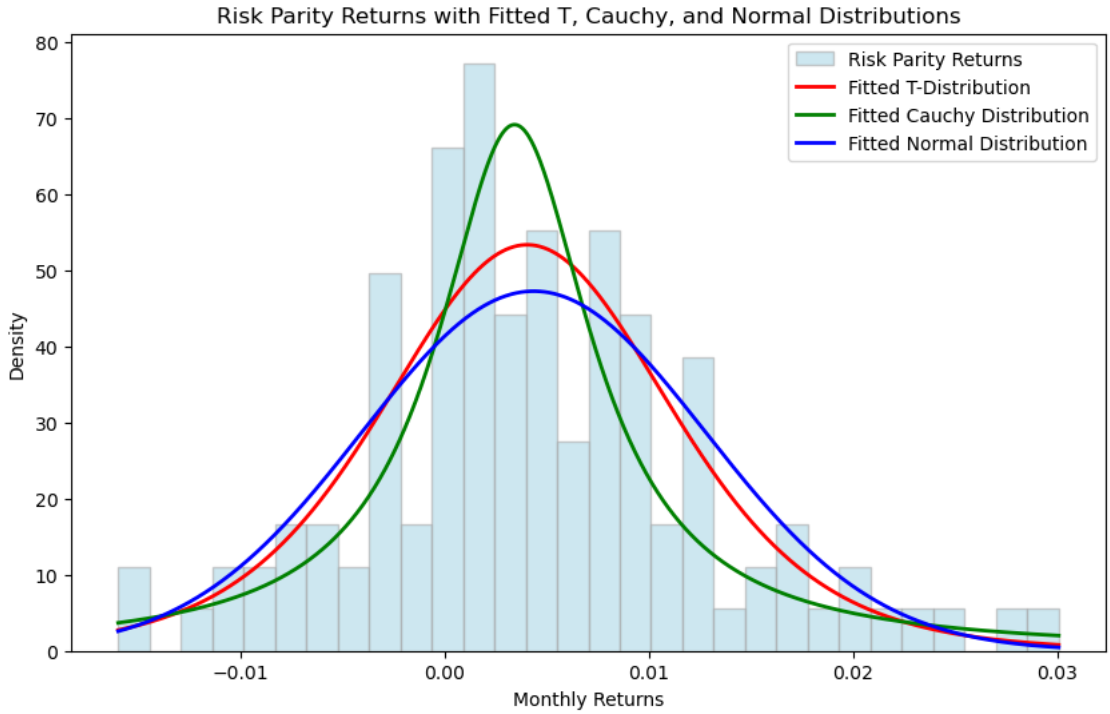
33 Raw Return Histograms of Optimized Portfolios with Fitted Probability Distributions (Not an Exhibit in Report)

[42]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import t, cauchy, norm

# performance_df contains 'Risk Parity', 'Minimum Variance', 'Maximum␣
 ↪Diversification', and 'VASIX (Benchmark)'
```

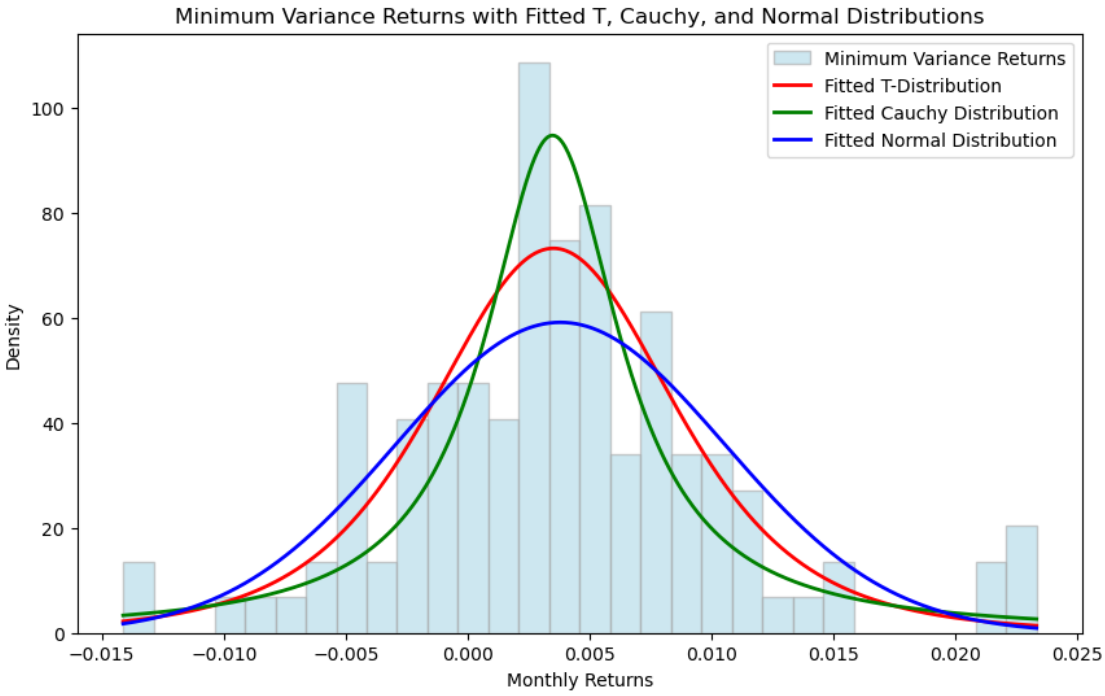
```
# List of portfolio names
portfolios = ['Risk Parity', 'Minimum Variance', 'Maximum Diversification']

# Function to plot histogram with fitted distributions
def plot_fitted_distributions(portfolio_name, returns):
    plt.figure(figsize=(10, 6))

    # Plot histogram of portfolio returns with dark grey borders
    plt.hist(returns, bins=30, density=True, alpha=0.6, color='lightblue',␣
 ↪edgecolor='darkgrey', label=f'{portfolio_name} Returns')

    # Fit T-distribution, Cauchy distribution, and Normal distribution
    params_t = t.fit(returns)
    params_cauchy = cauchy.fit(returns)
    params_norm = norm.fit(returns)

    # Generate points for plotting the fitted distributions
    x = np.linspace(min(returns), max(returns), 1000)

    # Plot fitted T-distribution
    plt.plot(x, t.pdf(x, *params_t), 'r-', lw=2, label='Fitted T-Distribution')

    # Plot fitted Cauchy distribution
    plt.plot(x, cauchy.pdf(x, *params_cauchy), 'g-', lw=2, label='Fitted Cauchy␣
 ↪Distribution')

    # Plot fitted Normal distribution
    plt.plot(x, norm.pdf(x, *params_norm), 'b-', lw=2, label='Fitted Normal␣
 ↪Distribution')

    # Add title and labels
    plt.title(f'{portfolio_name} Returns with Fitted T, Cauchy, and Normal␣
 ↪Distributions')
    plt.xlabel('Monthly Returns')
    plt.ylabel('Density')
    plt.legend()

    # Save the plot
    plt.savefig(f'{portfolio_name}_fitted_distributions.png',␣
 ↪bbox_inches='tight')

    # Optionally display the plot in Jupyter Notebook
    plt.show()

# Iterate over each portfolio and generate the histogram + fitted distributions
for portfolio in portfolios:
    print(f"Generating plot for {portfolio}...")
```

```
    portfolio_returns = performance_df[portfolio].dropna()
    plot_fitted_distributions(portfolio, portfolio_returns)
```

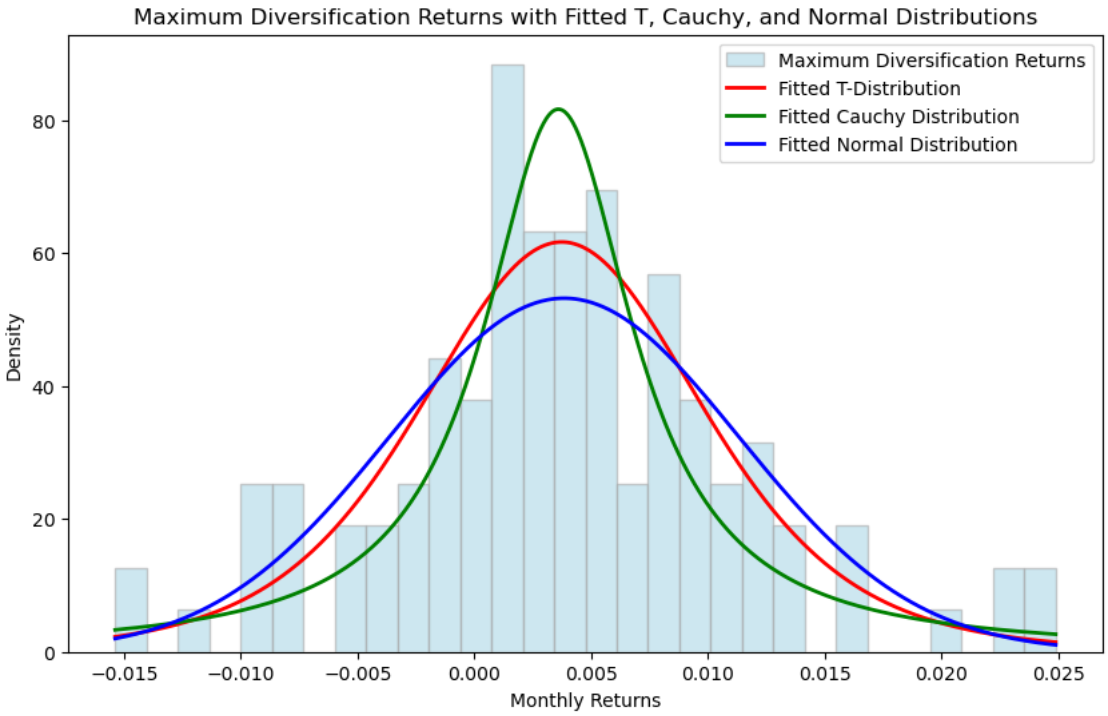
```
Generating plot for Risk Parity…
```




```
Generating plot for Minimum Variance…
```



```
Generating plot for Maximum Diversification…
```



34 Optimized Portfolio Quantitative Goodness of Fit Tests (AIC, BIC, K-S) (Table 17)

[104]:
```
import os
import numpy as np
import pandas as pd
import dataframe_image as dfi  # For exporting DataFrames as images
from scipy.stats import t, cauchy, norm, kstest

# Define the snippet name for output folder
snippet_name = "PORTFOLIO_AIC_BIC_KS_TABLE"

# Create the output folder
output_folder = os.path.join(os.getcwd(), snippet_name)
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# Function to compute AIC and BIC
def calculate_aic_bic(log_likelihood, num_params, num_data_points):
    aic = 2 * num_params - 2 * log_likelihood
    bic = np.log(num_data_points) * num_params - 2 * log_likelihood
    return aic, bic

# Initialize results list
results_list = []

# Portfolios and benchmark (use your performance_df generated from optimized␣
 ↪portfolios)
portfolios = ['Risk Parity', 'Minimum Variance', 'Maximum Diversification',␣
 ↪'VASIX (Benchmark)']

# Iterate over each portfolio
for portfolio in portfolios:
    returns = performance_df[portfolio].dropna()
    n = len(returns)

    # Fit T-Distribution
    df_t, loc_t, scale_t = t.fit(returns)
    ll_t = np.sum(np.log(t.pdf(returns, df_t, loc_t, scale_t)))
    aic_t, bic_t = calculate_aic_bic(ll_t, 3, n)
    ks_t = kstest(returns, 't', args=(df_t, loc_t, scale_t))

    # Fit Cauchy Distribution
    loc_cauchy, scale_cauchy = cauchy.fit(returns)
    ll_cauchy = np.sum(np.log(cauchy.pdf(returns, loc_cauchy, scale_cauchy)))
    aic_cauchy, bic_cauchy = calculate_aic_bic(ll_cauchy, 2, n)
    ks_cauchy = kstest(returns, 'cauchy', args=(loc_cauchy, scale_cauchy))
```

```
    # Fit Normal Distribution
    mu_normal, std_normal = norm.fit(returns)
    ll_normal = np.sum(np.log(norm.pdf(returns, mu_normal, std_normal)))
    aic_normal, bic_normal = calculate_aic_bic(ll_normal, 2, n)
    ks_normal = kstest(returns, 'norm', args=(mu_normal, std_normal))

    # Collect results for this portfolio
    results_list.append({
        'Portfolio': portfolio,
        'AIC_T': aic_t,
        'AIC_Cauchy': aic_cauchy,
        'AIC_Normal': aic_normal,
        'BIC_T': bic_t,
        'BIC_Cauchy': bic_cauchy,
        'BIC_Normal': bic_normal,
        'K-S_T': ks_t.statistic,
        'K-S_Cauchy': ks_cauchy.statistic,
        'K-S_Normal': ks_normal.statistic
    })

# Convert results to DataFrame
results_df = pd.DataFrame(results_list)

# Ensure the AIC and BIC columns are in the correct format
results_df[['AIC_T', 'AIC_Cauchy', 'AIC_Normal', 'BIC_T', 'BIC_Cauchy',␣
 ↪'BIC_Normal']] = \
    results_df[['AIC_T', 'AIC_Cauchy', 'AIC_Normal', 'BIC_T', 'BIC_Cauchy',␣
 ↪'BIC_Normal']].astype(float)

# Limit the display to 3 decimal places
pd.options.display.float_format = '{:.3f}'.format

# Function to highlight the best fit based on AIC, BIC, and K-S
def highlight_best_fit(row):
    aic_cols = ['AIC_T', 'AIC_Cauchy', 'AIC_Normal']
    bic_cols = ['BIC_T', 'BIC_Cauchy', 'BIC_Normal']
    ks_cols = ['K-S_T', 'K-S_Cauchy', 'K-S_Normal']

    min_aic = row[aic_cols].min()
    min_bic = row[bic_cols].min()
    min_ks = row[ks_cols].min()

    def highlight(val, col_type):
        if col_type == 'AIC' and val == min_aic:
            return 'background-color: lightgreen; font-weight: bold'
        elif col_type == 'BIC' and val == min_bic:
```

```
            return 'background-color: lightblue; font-weight: bold'
        elif col_type == 'K-S' and val == min_ks:
            return 'background-color: lightyellow; font-weight: bold'
        else:
            return ''

    return [
        highlight(val, 'AIC') if col in aic_cols else
        highlight(val, 'BIC') if col in bic_cols else
        highlight(val, 'K-S') if col in ks_cols else ''
        for col, val in row.items()
    ]

# Apply formatting to highlight the best fit for each metric group
formatted_results = results_df.style.apply(highlight_best_fit, axis=1).
 ↪format(precision=3)

# Save the formatted table as a PNG file inside the created folder
dfi.export(formatted_results, os.path.join(output_folder,␣
 ↪"portfolio_aic_bic_ks_table.png"))

# Optionally, display the formatted table in Jupyter
formatted_results
```

[104]: <pandas.io.formats.style.Styler at 0x15398c620>

35 Optimized Portfolio Monte Carlo Analysis (36-Months, 10,000 Iterations) (Table 18)

[52]:
```
import os
from PIL import Image
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import dataframe_image as dfi
from scipy.stats import t  # Import t-distribution
from IPython.display import display, Image as IPImage

# Define Monte Carlo simulation function for T-distribution
def monte_carlo_simulation(returns, portfolio_name, initial_balance=10000,␣
 ↪num_simulations=10000, months=36, seed=42, df=5):
    np.random.seed(seed)
    mean_monthly = returns.mean()
    std_monthly = returns.std()

    # Simulate using T-distribution
```

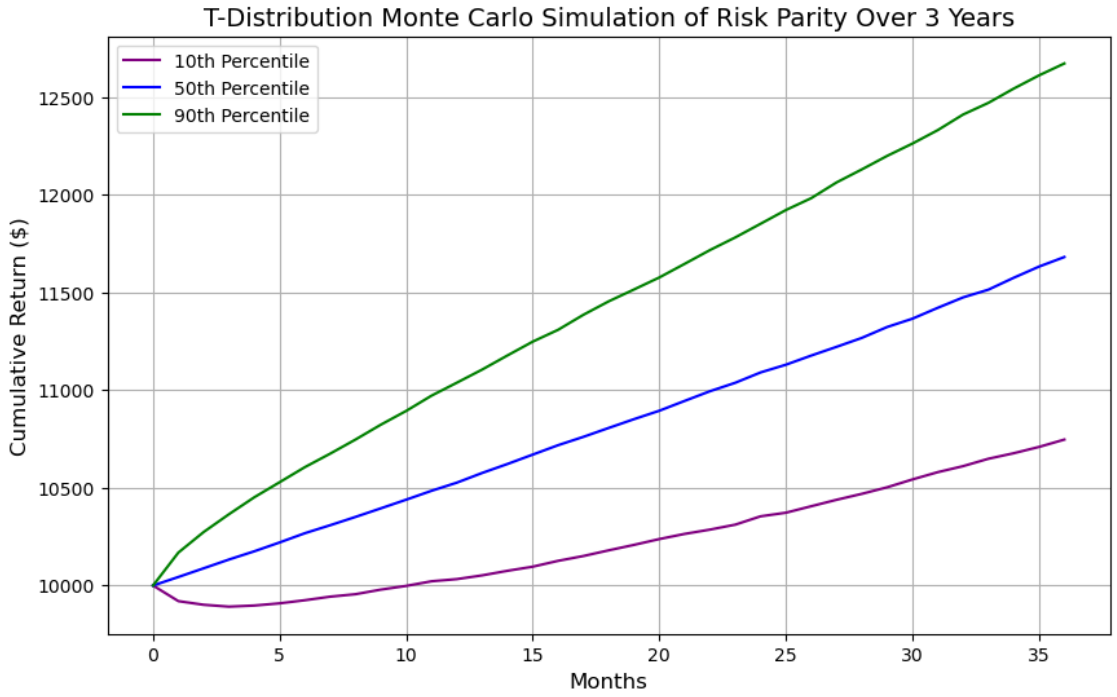
```
    simulations = t.rvs(df, loc=mean_monthly, scale=std_monthly,␣
↪size=(num_simulations, months))

    # Correct the calculation: Return series should be handled properly as␣
↪monthly returns
    cumulative_returns = np.cumprod(1 + simulations, axis=1) * initial_balance
    initial_balances = np.full((num_simulations, 1), initial_balance)
    cumulative_returns_with_zero = np.hstack((initial_balances,␣
↪cumulative_returns))

    # Portfolio statistics
    end_balances = np.percentile(cumulative_returns_with_zero[:, -1], [90, 50,␣
↪10])
    annual_compounded_return = (end_balances / initial_balance) ** (1 / (months␣
↪/ 12)) - 1

    # Simulated monthly returns for each simulation (i.e., relative changes for␣
↪monthly return calculation)
    returns_simulations = simulations  # These are the monthly returns already
    monthly_volatility = np.std(returns_simulations, axis=1)  # Std deviation␣
↪of monthly returns
    annualized_volatility = monthly_volatility * np.sqrt(12)  # Annualized␣
↪volatility

    # Max drawdown calculation using cumulative return paths
    max_drawdown = np.min(cumulative_returns_with_zero / np.maximum.
↪accumulate(cumulative_returns_with_zero, axis=1), axis=1) - 1

    # Create summary DataFrame with portfolio name as the first column
    summary = pd.DataFrame({
        "Portfolio": [portfolio_name] * 3,
        "Percentile": ["90th Percentile", "50th Percentile", "10th Percentile"],
        "Portfolio End Balance ($)": end_balances,
        "Annual Compounded Return (%)": (annual_compounded_return * 100),
        "Annualized Volatility (%)": np.percentile(annualized_volatility, [90,␣
↪50, 10]) * 100,
        "Maximum Drawdown (%)": np.percentile(max_drawdown, [90, 50, 10]) * 100
    })

    # Save individual plot and summary
    output_folder = f"TDIST_MONTE_CARLO_SIMULATIONS_ROLLING/{portfolio_name.
↪replace(' ', '_')}"
    os.makedirs(output_folder, exist_ok=True)
    plot_path = os.path.join(output_folder,␣
↪f"{portfolio_name}_monte_carlo_simulation_rolling.png")
    plt.figure(figsize=(10, 6), dpi=100)
```

```
    plt.plot(np.arange(0, months + 1), np.
↪percentile(cumulative_returns_with_zero, 10, axis=0), label='10th␣
↪Percentile', color='purple')
    plt.plot(np.arange(0, months + 1), np.
↪percentile(cumulative_returns_with_zero, 50, axis=0), label='50th␣
↪Percentile', color='blue')
    plt.plot(np.arange(0, months + 1), np.
↪percentile(cumulative_returns_with_zero, 90, axis=0), label='90th␣
↪Percentile', color='green')
    plt.title(f"T-Distribution Monte Carlo Simulation of {portfolio_name} Over␣
↪3 Years")
    plt.xlabel("Months")
    plt.ylabel("Cumulative Return ($)")
    plt.legend(loc='upper left')
    plt.grid(True)
    plt.savefig(plot_path, bbox_inches='tight')

    # Display the plot in Jupyter
    plt.show()  # Display the plot in Jupyter Notebook
    plt.close()

    summary_path = os.path.join(output_folder,␣
↪f"{portfolio_name}_summary_table_rolling.png")
    styled_summary = summary.style.format({
        "Portfolio End Balance ($)": "{:,.0f}",
        "Annual Compounded Return (%)": "{:.2f}",
        "Annualized Volatility (%)": "{:.2f}",
        "Maximum Drawdown (%)": "{:.2f}"
    })
    dfi.export(styled_summary, summary_path)

    # Display the summary image in Jupyter
    display(IPImage(summary_path))  # Display the summary image

    return plot_path, summary_path, summary  # Return paths to individual PNGs␣
↪and summary table

# Use rolling optimization portfolio data (from CODE SNIPPET 7)
portfolios = ['Risk Parity', 'Minimum Variance', 'Maximum Diversification',␣
↪'VASIX (Benchmark)']

# List to store file paths and summaries
all_plot_paths = []
all_summary_dfs = []

# Iterate over each portfolio and run the Monte Carlo simulation
```

```
for portfolio in portfolios:
    print(f"Running Rolling Monte Carlo Simulation for {portfolio}")
    portfolio_returns = performance_df[portfolio].dropna()  # Use rolling␣
 ↪portfolio returns from performance_df
    plot_path, summary_path, summary_df =␣
 ↪monte_carlo_simulation(portfolio_returns, portfolio, df=5)
    all_plot_paths.append(plot_path)
    all_summary_dfs.append(summary_df)

# Combine all summary tables into a single DataFrame
combined_summary_df = pd.concat(all_summary_dfs).reset_index(drop=True)

# Function to apply a thinner black line after every portfolio block of 3 rows
def highlight_portfolio_change(df):
    styles = pd.DataFrame('', index=df.index, columns=df.columns)

    # Add the line after every 3rd row corresponding to each portfolio
    for i in range(2, len(df), 3):  # Start from the third row and step by 3
        styles.loc[i, :] = 'border-bottom: 1px solid black'  # Thinner black␣
 ↪border after each portfolio block

    return styles

# Apply the highlighting function to create thinner lines between portfolio␣
 ↪blocks
styled_combined_summary = combined_summary_df.style.
 ↪apply(highlight_portfolio_change, axis=None).format({
    "Portfolio End Balance ($)": "{:,.0f}",
    "Annual Compounded Return (%)": "{:.2f}",
    "Annualized Volatility (%)": "{:.2f}",
    "Maximum Drawdown (%)": "{:.2f}"
}).set_caption("Monte Carlo Simulation Summary for Rolling Portfolios")

# Save the styled combined summary table with thinner portfolio separators
combined_summary_path = os.path.join("TDIST_MONTE_CARLO_SIMULATIONS_ROLLING",␣
 ↪"combined_summary_table_rolling_with_thinner_lines.png")
dfi.export(styled_combined_summary, combined_summary_path)

# Display the combined summary in Jupyter Notebook
display(IPImage(combined_summary_path))  # Display combined summary image

# Function to combine images in 2x2 layout for plots
def combine_images_2x2(image_paths, output_file):
    images = [Image.open(image_path) for image_path in image_paths]
    widths, heights = zip(*(i.size for i in images))
```

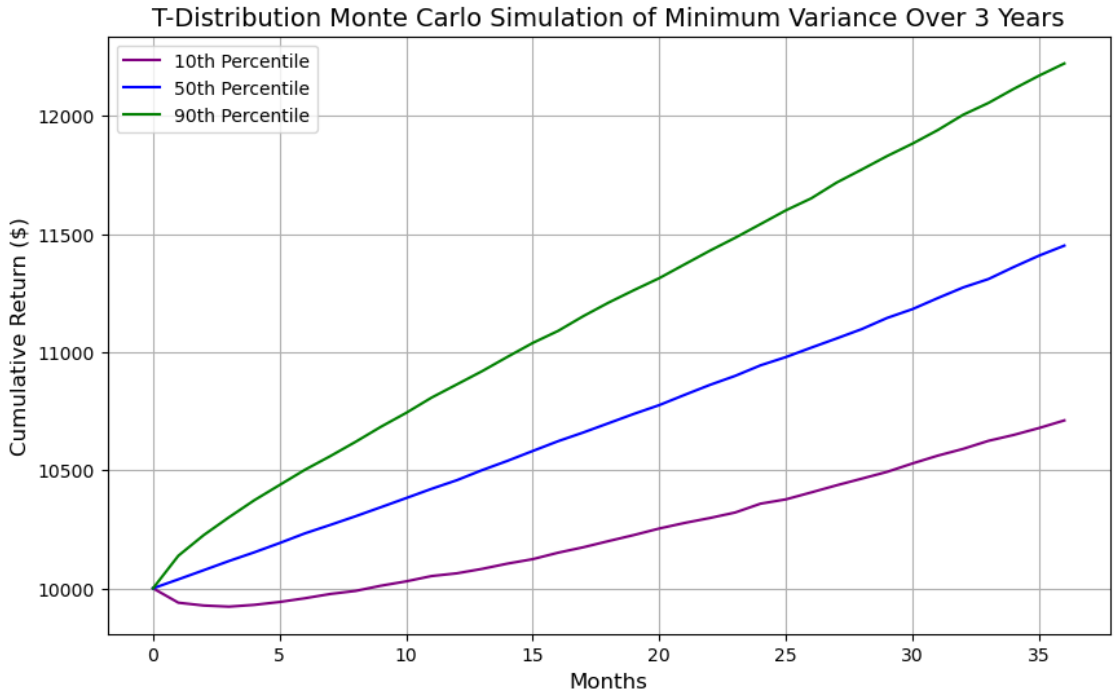
```
    max_width = max(widths)
    max_height = max(heights)

    # Create a new image with 2x2 grid
    combined_image = Image.new('RGB', (2 * max_width, 2 * max_height))

    # Paste images into a 2x2 grid
    combined_image.paste(images[0], (0, 0))
    combined_image.paste(images[1], (max_width, 0))
    combined_image.paste(images[2], (0, max_height))
    combined_image.paste(images[3], (max_width, max_height))

    combined_image.save(output_file)

# Combine all individual plot PNGs into one 2x2 image
combined_plot_path = os.path.join("TDIST_MONTE_CARLO_SIMULATIONS_ROLLING",␣
 ↪"combined_monte_carlo_plots_2x2_rolling.png")
combine_images_2x2(all_plot_paths, combined_plot_path)

# Display the combined plot image in Jupyter Notebook
display(IPImage(combined_plot_path))  # Display combined plot image

# Output paths
print(f"Combined rolling Monte Carlo plots saved to: {combined_plot_path}")
print(f"Combined rolling summaries with thinner lines saved to:␣
 ↪{combined_summary_path}")
```

```
Running Rolling Monte Carlo Simulation for Risk Parity
```



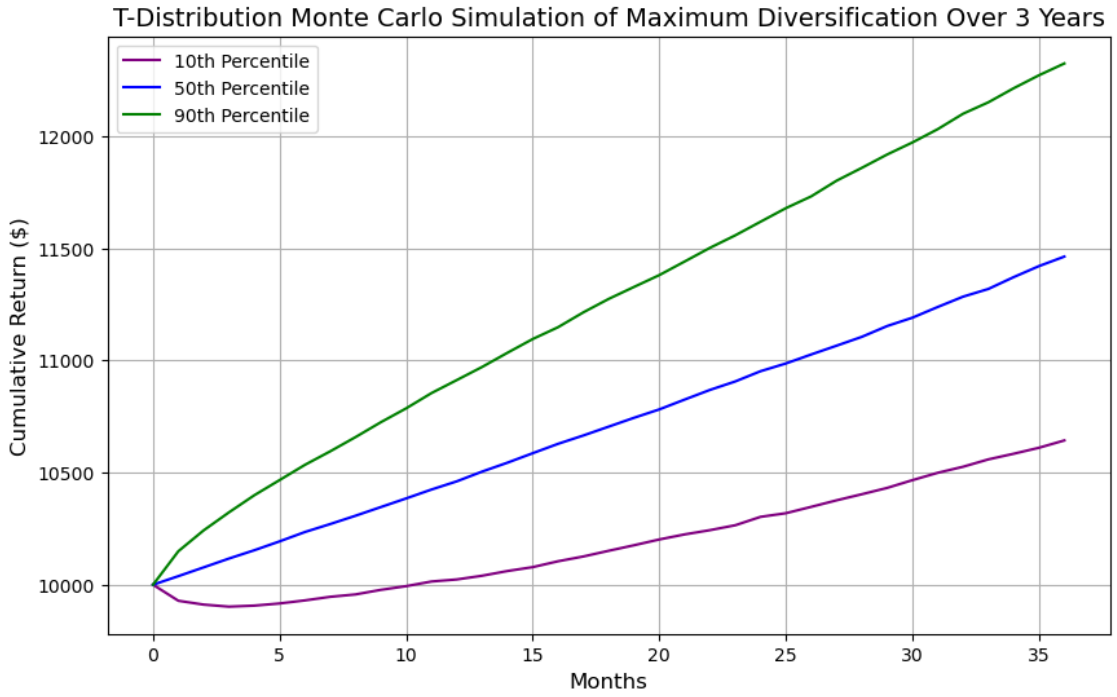
	Portfolio	Percentile	Portfolio End Balance ($)	Annual Compounded Return (%)	Annualized Volatility (%)	Maximum Drawdown (%)
0	Risk Parity	90th Percentile	12,672	8.21	4.54	-1.44
1	Risk Parity	50th Percentile	11,681	5.32	3.58	-2.74
2	Risk Parity	10th Percentile	10,747	2.43	2.89	-5.38

Running Rolling Monte Carlo Simulation for Minimum Variance



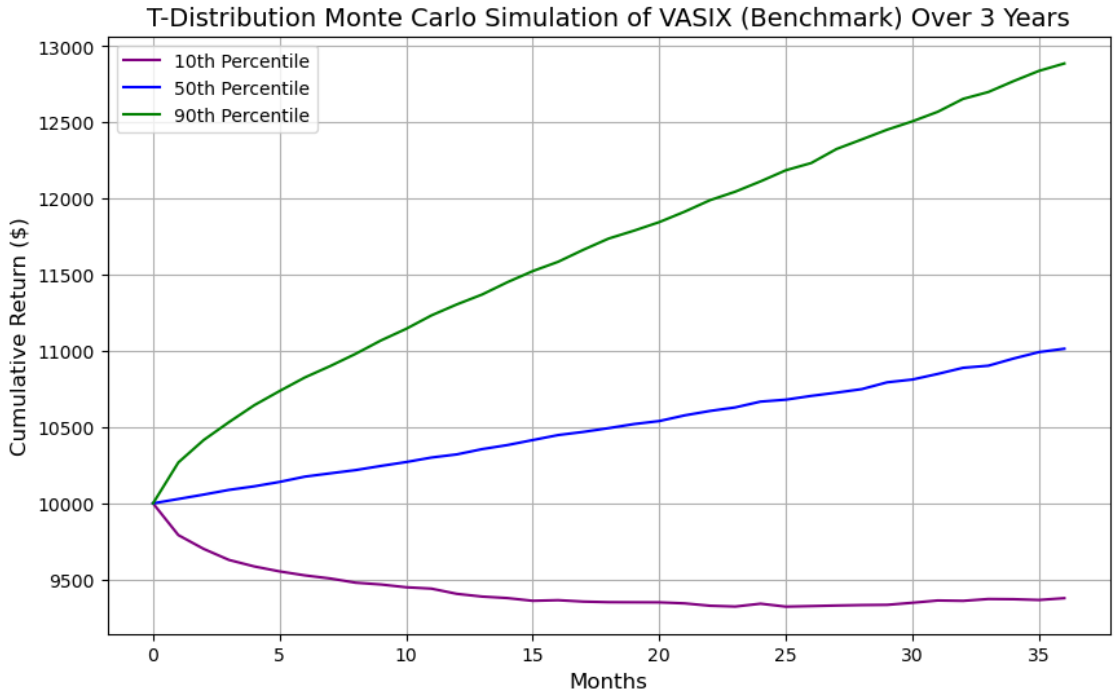
	Portfolio	Percentile	Portfolio End Balance ($)	Annual Compounded Return (%)	Annualized Volatility (%)	Maximum Drawdown (%)
0	Minimum Variance	90th Percentile	12,222	6.92	3.63	-1.09
1	Minimum Variance	50th Percentile	11,451	4.62	2.87	-2.09
2	Minimum Variance	10th Percentile	10,711	2.32	2.31	-4.13

Running Rolling Monte Carlo Simulation for Maximum Diversification



	Portfolio	Percentile	Portfolio End Balance ($)	Annual Compounded Return (%)	Annualized Volatility (%)	Maximum Drawdown (%)
0	Maximum Diversification	90th Percentile	12,324	7.21	4.03	-1.29
1	Maximum Diversification	50th Percentile	11,463	4.66	3.19	-2.46
2	Maximum Diversification	10th Percentile	10,643	2.10	2.57	-4.83

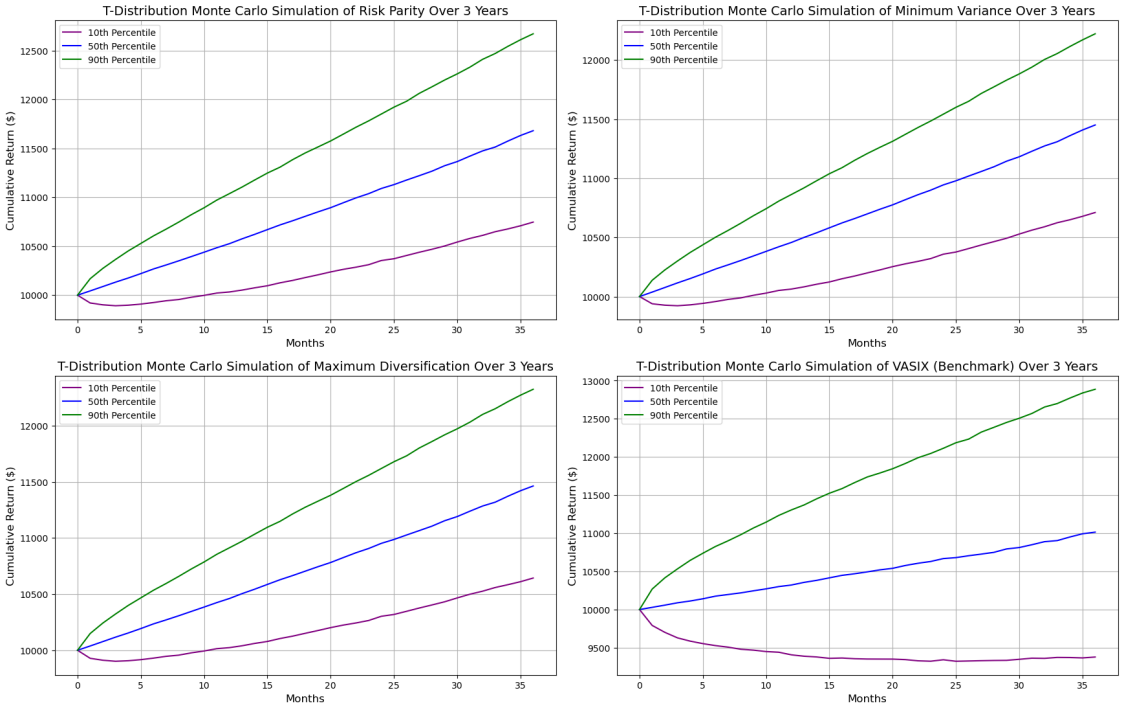
Running Rolling Monte Carlo Simulation for VASIX (Benchmark)



	Portfolio	Percentile	Portfolio End Balance ($)	Annual Compounded Return (%)	Annualized Volatility (%)	Maximum Drawdown (%)
0	VASIX (Benchmark)	90th Percentile	12,887	8.82	8.71	-4.41
1	VASIX (Benchmark)	50th Percentile	11,015	3.27	6.88	-8.18
2	VASIX (Benchmark)	10th Percentile	9,378	-2.12	5.55	-15.54

Monte Carlo Simulation Summary for Rolling Portfolios

	Portfolio	Percentile	Portfolio End Balance ($)	Annual Compounded Return (%)	Annualized Volatility (%)	Maximum Drawdown (%)
0	Risk Parity	90th Percentile	12,672	8.21	4.54	-1.44
1	Risk Parity	50th Percentile	11,681	5.32	3.58	-2.74
2	Risk Parity	10th Percentile	10,747	2.43	2.89	-5.38
3	Minimum Variance	90th Percentile	12,222	6.92	3.63	-1.09
4	Minimum Variance	50th Percentile	11,451	4.62	2.87	-2.09
5	Minimum Variance	10th Percentile	10,711	2.32	2.31	-4.13
6	Maximum Diversification	90th Percentile	12,324	7.21	4.03	-1.29
7	Maximum Diversification	50th Percentile	11,463	4.66	3.19	-2.46
8	Maximum Diversification	10th Percentile	10,643	2.10	2.57	-4.83
9	VASIX (Benchmark)	90th Percentile	12,887	8.82	8.71	-4.41
10	VASIX (Benchmark)	50th Percentile	11,015	3.27	6.88	-8.18
11	VASIX (Benchmark)	10th Percentile	9,378	-2.12	5.55	-15.54




```
Combined rolling Monte Carlo plots saved to:
TDIST_MONTE_CARLO_SIMULATIONS_ROLLING/combined_monte_carlo_plots_2x2_rolling.png
Combined rolling summaries with thinner lines saved to: TDIST_MONTE_CARLO_SIMULA
TIONS_ROLLING/combined_summary_table_rolling_with_thinner_lines.png
```

36 Rolling Optimized Portfolio Annual Return and Performance Summary Table Versus Benchmark (Table 19)

[44]:
```python
import pandas as pd
import numpy as np

# Assuming 'performance_df' already has the monthly returns of all portfolios
# Shift data by one month to ensure no look-ahead bias (lag by 1 month)
performance_df_lagged = performance_df.shift(1)

# Calculate Annual Returns for the Rolling Optimized Portfolios and VASIX
def calculate_annual_returns(df, column_name):
    """Calculate annual returns for the portfolio."""
    annual_returns = df[column_name].resample('YE').apply(lambda x: (1 + x).
 ↪prod() - 1)
    return (annual_returns * 100).round(2)  # Convert to percentage and round

# Build Annual Returns Table (starting after the rolling window)
```

```
def build_annual_returns_table(performance_df, start_date):
    """Create a table for annual returns starting from the given start date."""
    annual_returns_rp = calculate_annual_returns(performance_df, 'Risk␣
 ↪Parity')[start_date:]
    annual_returns_mv = calculate_annual_returns(performance_df, 'Minimum␣
 ↪Variance')[start_date:]
    annual_returns_md = calculate_annual_returns(performance_df, 'Maximum␣
 ↪Diversification')[start_date:]
    annual_returns_vasix = calculate_annual_returns(performance_df, 'VASIX␣
 ↪(Benchmark)')[start_date:]

    annual_return_table = pd.DataFrame({
        'Risk Parity (%)': annual_returns_rp,
        'Minimum Variance (%)': annual_returns_mv,
        'Maximum Diversification (%)': annual_returns_md,
        'VASIX (Benchmark) (%)': annual_returns_vasix
    })
    return annual_return_table

# Build Performance Metrics Table
def calculate_cagr(df, column_name):
    """Calculate the Compound Annual Growth Rate (CAGR)."""
    days_diff = (df.index[-1] - df.index[0]).days
    years = days_diff / 365.25
    cumulative_return = (1 + df[column_name]).prod()
    return (cumulative_return ** (1 / years) - 1) * 100  # Convert to percentage

def calculate_max_drawdown(df, column_name):
    """Calculate the Maximum Drawdown."""
    cumulative_return = (1 + df[column_name]).cumprod()
    rolling_max = cumulative_return.cummax()
    drawdown = (cumulative_return - rolling_max) / rolling_max
    return drawdown.min() * 100  # Convert to percentage

def build_performance_metrics_table(df):
    """Create a table for performance metrics."""
    metrics = pd.DataFrame({
        'CAGR (%)': [calculate_cagr(df, col) for col in df.columns],
        'Standard Deviation (%)': [df[col].std() * np.sqrt(12) * 100 for col in␣
 ↪df.columns],
        'Maximum Drawdown (%)': [calculate_max_drawdown(df, col) for col in df.
 ↪columns]
    }, index=df.columns)
    return metrics.round(2)
```

```
# Get the start date for rolling portfolio returns (12 months after the␣
 ↪earliest date)
start_date = performance_df.index[12]  # Assuming monthly data and a 12-month␣
 ↪rolling window

# Remove the time portion from the start date printout
formatted_start_date = start_date.strftime('%Y-%m-%d')
print(f"First date for rolling portfolio returns: {formatted_start_date}")

# Adjust settings to prevent line breaks
pd.set_option('display.expand_frame_repr', False)  # Prevent line breaking for␣
 ↪large tables
pd.set_option('display.max_columns', None)  # Show all columns

# Create the tables (starting from the start date)
annual_return_table = build_annual_returns_table(performance_df_lagged,␣
 ↪start_date)
performance_metrics_table =␣
 ↪build_performance_metrics_table(performance_df_lagged[start_date:])

# Display the tables in the format you want
print("\nAnnual Return Table (%):")
print(annual_return_table)

print("\nPerformance Metrics Table:")
print(performance_metrics_table)
```

```
First date for rolling portfolio returns: 2015-11-30

Annual Return Table (%):
            Risk Parity (%)  Minimum Variance (%)  Maximum Diversification (%)
VASIX (Benchmark) (%)
Date
2015-12-31             2.50                  2.71                         2.67
0.96
2016-12-31             2.95                  2.32                         1.87
3.13
2017-12-31             3.97                  0.88                         1.35
7.06
2018-12-31             0.05                  2.97                         1.91
-0.37
2019-12-31             7.33                  6.42                         7.19
11.35
2020-12-31             6.10                  6.16                         7.48
8.35
2021-12-31             9.23                  7.09                         8.05
2.80
```

```
2022-12-31             7.79                  5.70                        5.70
-11.94
2023-12-31             4.13                  3.46                        1.99
3.28
2024-12-31             7.25                  6.68                        6.60
7.83

Performance Metrics Table:
                        CAGR (%)  Standard Deviation (%)  Maximum Drawdown (%)
Risk Parity                 5.59                    2.81                 -1.62
Minimum Variance            4.79                    2.20                 -1.52
Maximum Diversification     4.88                    2.50                 -2.00
VASIX (Benchmark)           3.55                    5.85                -16.72
```

Project GitHub Repository: https://github.com/littlecl42/AAA500_FinalProject_CL_IL/

Carrie Final Project Data Analysis

October 17, 2024

1 Opportunity dataset

Carrie Little

1.0.1 Import Necessary Libraries

[1]:
```
# Import All Necessary Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
import cvxpy as cp
```

1.0.2 Load Opportunity Dataset

[2]:
```
# Load Opportunity Dataset
df = pd.read_csv('Opportunity_Set.csv', parse_dates=['Date'], index_col='Date')␣
 ↪     # Load Dataset as Dataframe
print(df.head())                                                                ␣
 ↪     # Diaplay 1st 5 inDataframe
```

```
            Vanguard LifeStrategy Income Fund (VASIX)  \
Date
2014-11-30                                     0.0094
2014-12-31                                    -0.0005
2015-01-31                                     0.0141
2015-02-28                                     0.0033
2015-03-31                                     0.0018

            Vanguard Total World Stock ETF (VT)  \
Date
2014-11-30                               0.0126
2014-12-31                              -0.0199
2015-01-31                              -0.0163
2015-02-28                               0.0595
2015-03-31                              -0.0121
```

Date	PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)	AQR Diversified Arbitrage I (ADAIX)	iShares Gold Trust (IAU)	Bitcoin Market Price USD (^BTC)	AQR Risk-Balanced Commodities Strategy I (ARCIX)	AQR Long-Short Equity I (QLEIX)	AQR Style Premia Alternative I (QSPIX)
2014-11-30	0.0414	-0.0066	-0.0053	0.0969	-0.0726	0.0248	0.0412
2014-12-31	0.0612	-0.0117	0.0133	-0.1777	-0.0412	0.0140	0.0002
2015-01-31	0.1600	-0.0059	0.0865	-0.2677	-0.0287	0.0156	-0.0112
2015-02-28	-0.1007	0.0069	-0.0579	0.1062	0.0044	0.0236	-0.0390
2015-03-31	0.0117	0.0000	-0.0222	-0.0150	-0.0573	-0.0027	0.0256

```
            AQR Equity Market Neutral I (QMNIX)  \
Date
2014-11-30                               0.0257
2014-12-31                               0.0195
2015-01-31                               0.0290
2015-02-28                              -0.0078
2015-03-31                               0.0049

            AQR Macro Opportunities I (QGMIX)  \
Date
2014-11-30                              0.0154
2014-12-31                              0.0039
2015-01-31                             -0.0070
2015-02-28                              0.0091
2015-03-31                              0.0261

            AGF U.S. Market Neutral Anti-Beta (BTAL)  \
Date
2014-11-30                                  0.0235
2014-12-31                                  0.0294
2015-01-31                                  0.0320
2015-02-28                                 -0.0568
2015-03-31                                  0.0000

            AQR Managed Futures Strategy HV I (QMHIX)  \
Date
2014-11-30                                   0.1159
2014-12-31                                   0.0461
2015-01-31                                   0.0721
2015-02-28                                  -0.0108
2015-03-31                                   0.0655

            Invesco DB US Dollar Bullish (UUP)  \
Date
2014-11-30                             0.0165
2014-12-31                             0.0213
2015-01-31                             0.0484
2015-02-28                             0.0028
2015-03-31                             0.0278

            ProShares VIX Mid-Term Futures (VIXM)
Date
2014-11-30                               -0.0298
2014-12-31                                0.0553
2015-01-31                                0.0762
2015-02-28                               -0.1145
2015-03-31                                0.0033
```

1.0.3 Dataframe Info

[3]:
```
# Dataframe Info
df.info()
    # General information about the dataset
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 118 entries, 2014-11-30 to 2024-08-31
Data columns (total 15 columns):
 #   Column                                          Non-Null Count  Dtype
---  ------                                          --------------  -----
 0   Vanguard LifeStrategy Income Fund (VASIX)       118 non-null    float64
 1   Vanguard Total World Stock ETF (VT)             118 non-null    float64
 2   PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)    118 non-null    float64
 3   AQR Diversified Arbitrage I (ADAIX)             118 non-null    float64
 4   iShares Gold Trust (IAU)                        118 non-null    float64
 5   Bitcoin Market Price USD (^BTC)                 118 non-null    float64
 6   AQR Risk-Balanced Commodities Strategy I (ARCIX) 118 non-null   float64
 7   AQR Long-Short Equity I (QLEIX)                 118 non-null    float64
 8   AQR Style Premia Alternative I (QSPIX)          118 non-null    float64
 9   AQR Equity Market Neutral I (QMNIX)             118 non-null    float64
 10  AQR Macro Opportunities I (QGMIX)               118 non-null    float64
 11  AGF U.S. Market Neutral Anti-Beta (BTAL)        118 non-null    float64
 12  AQR Managed Futures Strategy HV I (QMHIX)       118 non-null    float64
 13  Invesco DB US Dollar Bullish (UUP)              118 non-null    float64
 14  ProShares VIX Mid-Term Futures (VIXM)           118 non-null    float64
dtypes: float64(15)
memory usage: 14.8 KB
```

1.0.4 Descriptive statistics

[4]:
```
# Descriptive Statistics
# Describe
summary_stats = df.describe()
    # Create a Dataframe of Descriptive Statistics
summary_stats
    # Display the Dataframe of Descriptive Statistics
```

[4]:
```
       Vanguard LifeStrategy Income Fund (VASIX)  \
count                                 118.000000
mean                                    0.002883
std                                     0.016261
min                                    -0.049200
25%                                    -0.002900
50%                                     0.004700
75%                                     0.009700
max                                     0.050400
```

	Vanguard Total World Stock ETF (VT)
count	118.000000
mean	0.008376
std	0.043483
min	-0.147600
25%	-0.018850
50%	0.012250
75%	0.030525
max	0.123700

	PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)
count	118.000000
mean	0.001381
std	0.060799
min	-0.134300
25%	-0.041300
50%	0.000800
75%	0.037725
max	0.175900

	AQR Diversified Arbitrage I (ADAIX)	iShares Gold Trust (IAU)
count	118.000000	118.000000
mean	0.004370	0.007036
std	0.016923	0.040440
min	-0.081500	-0.083600
25%	-0.003275	-0.019725
50%	0.004300	-0.000400
75%	0.009775	0.029250
max	0.074300	0.112200

	Bitcoin Market Price USD (^BTC)
count	118.000000
mean	0.066718
std	0.221176
min	-0.406000
25%	-0.080100
50%	0.046800
75%	0.182350
max	0.720000

	AQR Risk-Balanced Commodities Strategy I (ARCIX)
count	118.000000
mean	0.005475
std	0.046905
min	-0.144200
25%	-0.022500
50%	0.001600

```
75%                                          0.039500
max                                          0.121700

       AQR Long-Short Equity I (QLEIX)  \
count                       118.000000
mean                          0.008895
std                           0.033046
min                          -0.082100
25%                          -0.010925
50%                           0.009150
75%                           0.025425
max                           0.115800

       AQR Style Premia Alternative I (QSPIX)  \
count                              118.000000
mean                                 0.005400
std                                  0.038787
min                                 -0.078100
25%                                 -0.014750
50%                                  0.000000
75%                                  0.017450
max                                  0.140800

       AQR Equity Market Neutral I (QMNIX)  AQR Macro Opportunities I (QGMIX)  \
count                           118.000000                          118.000000
mean                              0.005330                            0.002896
std                               0.028727                            0.021807
min                              -0.061400                           -0.071200
25%                              -0.010275                           -0.008550
50%                               0.001550                            0.002150
75%                               0.020950                            0.015050
max                               0.110700                            0.066800

       AGF U.S. Market Neutral Anti-Beta (BTAL)  \
count                                118.000000
mean                                   0.001880
std                                    0.042857
min                                   -0.149600
25%                                   -0.021775
50%                                   -0.000650
75%                                    0.026825
max                                    0.094800

       AQR Managed Futures Strategy HV I (QMHIX)  \
count                                118.000000
mean                                   0.003564
std                                    0.047422
```

```
min                                    -0.088500
25%                                    -0.030075
50%                                    -0.001050
75%                                     0.036000
max                                     0.127500

       Invesco DB US Dollar Bullish (UUP)  \
count                          118.000000
mean                             0.002768
std                              0.019131
min                             -0.047300
25%                             -0.013000
50%                              0.004000
75%                              0.016425
max                              0.048400

       ProShares VIX Mid-Term Futures (VIXM)
count                             118.000000
mean                               -0.008486
std                                 0.093629
min                                -0.180100
25%                                -0.059250
50%                                -0.018150
75%                                 0.020825
max                                 0.628900
```

[5]:
```
# Display Skew and Kurtosis and Descriptives
from scipy.stats import skew, kurtosis
skewness = skew(df)                                                    #␣
 ↪Create an Array of Skewness of each asset
kurt = kurtosis(df)                                                    #␣
 ↪Create an Array of Kurtosis of each asset

# Creating a dataframe to display the results
results_df = pd.DataFrame({
    'Mean' : df.mean(),
    'Median' : df.median(),
    'Standard Deviation' : df.std(),
    'Skewness': skewness,
    'Kurtosis': kurt
})

results_df
```

[5]:
```
                                                 Mean   Median  \
Vanguard LifeStrategy Income Fund (VASIX)    0.002883  0.00470
Vanguard Total World Stock ETF (VT)          0.008376  0.01225
```

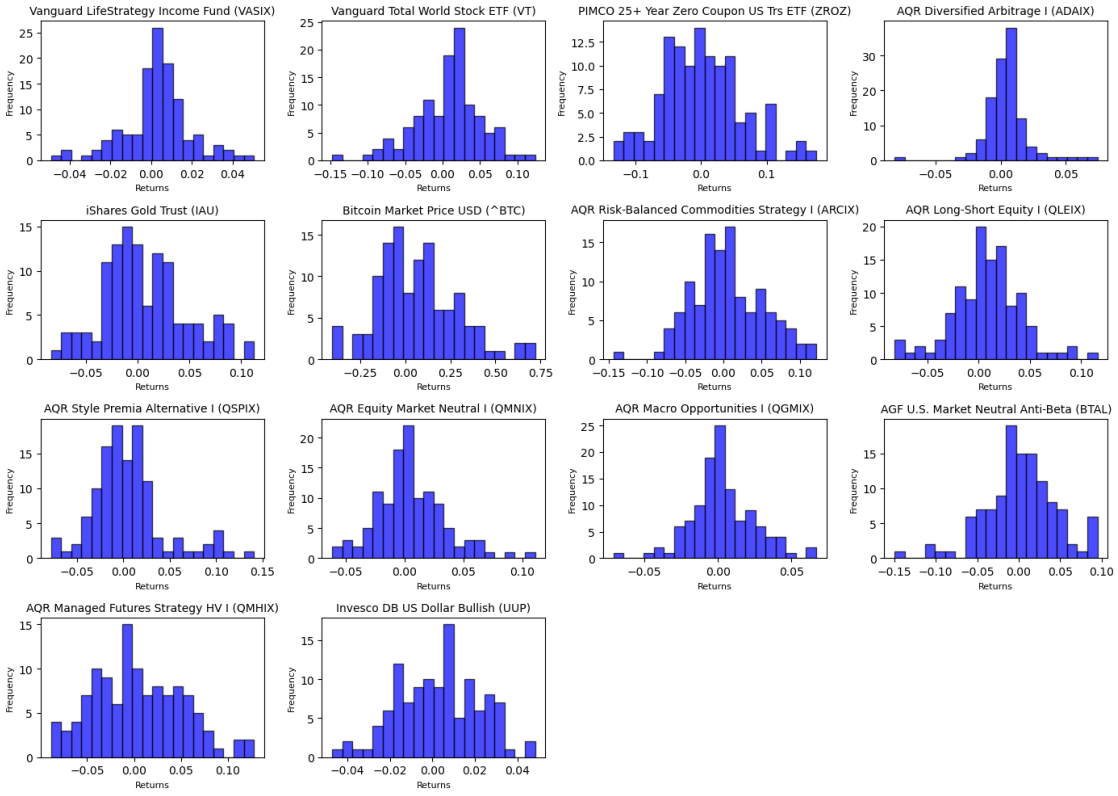
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)	0.001381	0.00080
AQR Diversified Arbitrage I (ADAIX)	0.004370	0.00430
iShares Gold Trust (IAU)	0.007036	-0.00040
Bitcoin Market Price USD (^BTC)	0.066718	0.04680
AQR Risk-Balanced Commodities Strategy I (ARCIX)	0.005475	0.00160
AQR Long-Short Equity I (QLEIX)	0.008895	0.00915
AQR Style Premia Alternative I (QSPIX)	0.005400	0.00000
AQR Equity Market Neutral I (QMNIX)	0.005330	0.00155
AQR Macro Opportunities I (QGMIX)	0.002896	0.00215
AGF U.S. Market Neutral Anti-Beta (BTAL)	0.001880	-0.00065
AQR Managed Futures Strategy HV I (QMHIX)	0.003564	-0.00105
Invesco DB US Dollar Bullish (UUP)	0.002768	0.00400
ProShares VIX Mid-Term Futures (VIXM)	-0.008486	-0.01815

	Standard Deviation	\
Vanguard LifeStrategy Income Fund (VASIX)	0.016261	
Vanguard Total World Stock ETF (VT)	0.043483	
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)	0.060799	
AQR Diversified Arbitrage I (ADAIX)	0.016923	
iShares Gold Trust (IAU)	0.040440	
Bitcoin Market Price USD (^BTC)	0.221176	
AQR Risk-Balanced Commodities Strategy I (ARCIX)	0.046905	
AQR Long-Short Equity I (QLEIX)	0.033046	
AQR Style Premia Alternative I (QSPIX)	0.038787	
AQR Equity Market Neutral I (QMNIX)	0.028727	
AQR Macro Opportunities I (QGMIX)	0.021807	
AGF U.S. Market Neutral Anti-Beta (BTAL)	0.042857	
AQR Managed Futures Strategy HV I (QMHIX)	0.047422	
Invesco DB US Dollar Bullish (UUP)	0.019131	
ProShares VIX Mid-Term Futures (VIXM)	0.093629	

	Skewness	Kurtosis
Vanguard LifeStrategy Income Fund (VASIX)	-0.240603	1.472664
Vanguard Total World Stock ETF (VT)	-0.426024	0.928619
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)	0.378626	0.262557
AQR Diversified Arbitrage I (ADAIX)	0.089222	8.083715
iShares Gold Trust (IAU)	0.374345	-0.113385
Bitcoin Market Price USD (^BTC)	0.599982	0.526931
AQR Risk-Balanced Commodities Strategy I (ARCIX)	0.105015	0.078234
AQR Long-Short Equity I (QLEIX)	0.016090	1.162428
AQR Style Premia Alternative I (QSPIX)	1.015599	1.697636
AQR Equity Market Neutral I (QMNIX)	0.637322	1.282440
AQR Macro Opportunities I (QGMIX)	0.102938	1.169489
AGF U.S. Market Neutral Anti-Beta (BTAL)	-0.365865	0.913209
AQR Managed Futures Strategy HV I (QMHIX)	0.294539	-0.303861
Invesco DB US Dollar Bullish (UUP)	-0.079627	-0.337540
ProShares VIX Mid-Term Futures (VIXM)	2.869086	17.097321

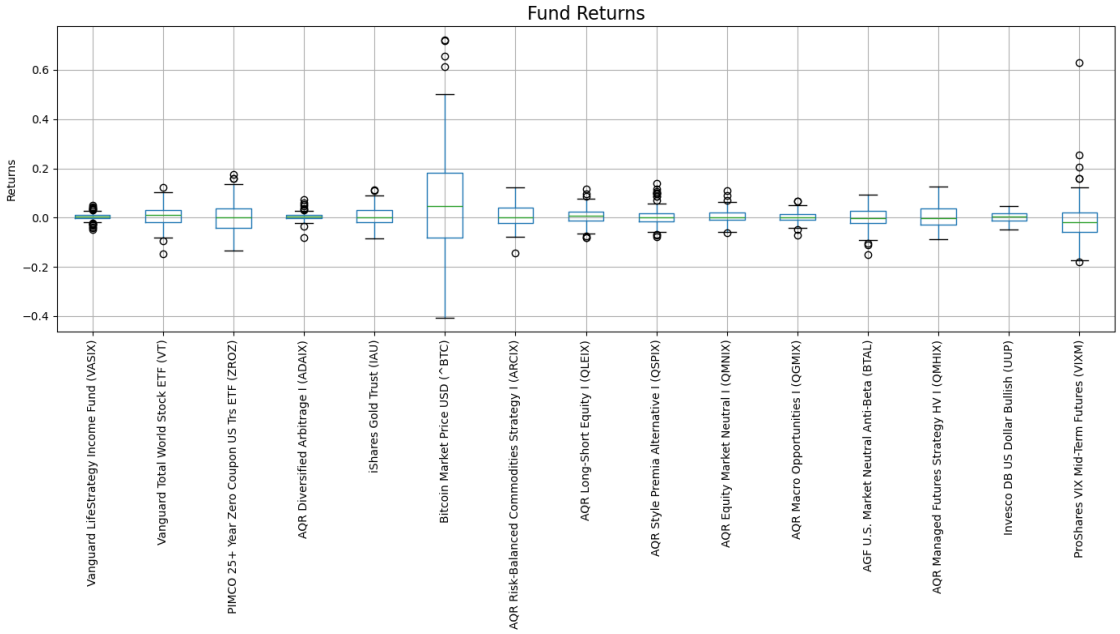
1.0.5 Histogram

[6]:
```
# Descriptive Plots
# Histogram
plt.figure(figsize=(14, 10))

# Iterate through each fund and create a histogram
for i, column in enumerate(df.columns[:-1], 1):  # Exclude the last␣
 ↪'DBSCAN_Cluster' column
    plt.subplot(4, 4, i)  # Creating a grid of subplots
    plt.hist(df[column], bins=20, alpha=0.7, color='blue', edgecolor='black')
    plt.title(column, fontsize=10)
    plt.xlabel('Returns', fontsize=8)
    plt.ylabel('Frequency', fontsize=8)

plt.tight_layout()
plt.show()
```



1.0.6 Boxplot

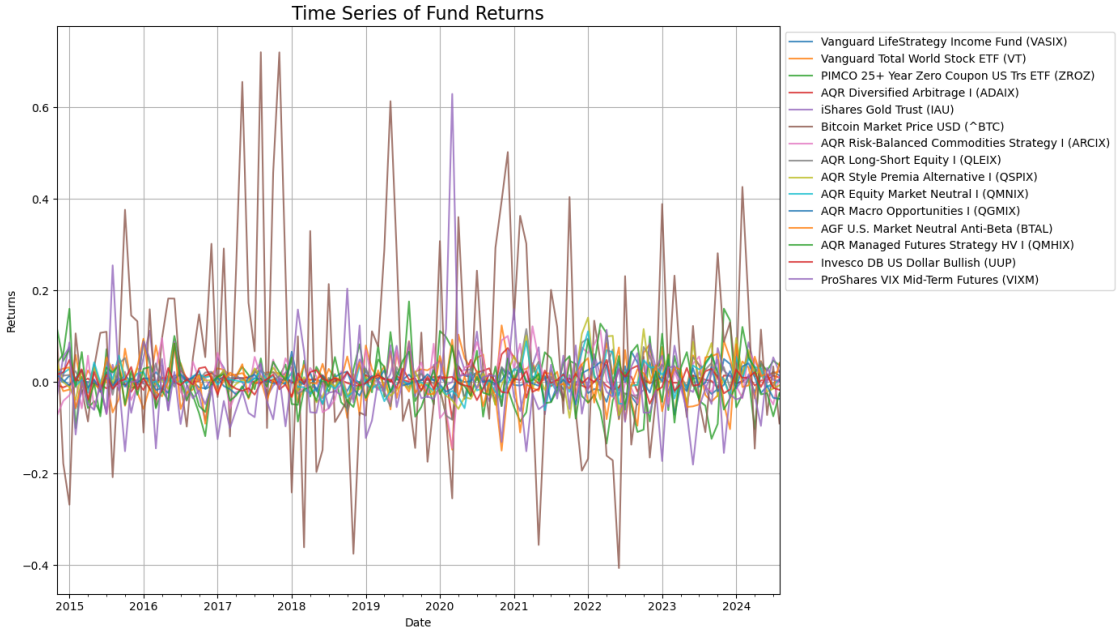
[7]:
```
# Descriptive Plots
# Boxplot
plt.figure(figsize=(14, 8))

df.boxplot()
plt.title('Fund Returns', fontsize=16)
plt.ylabel('Returns')
plt.xticks(rotation=90)
plt.grid(True)

# Show the plot
plt.tight_layout()
plt.show()
```



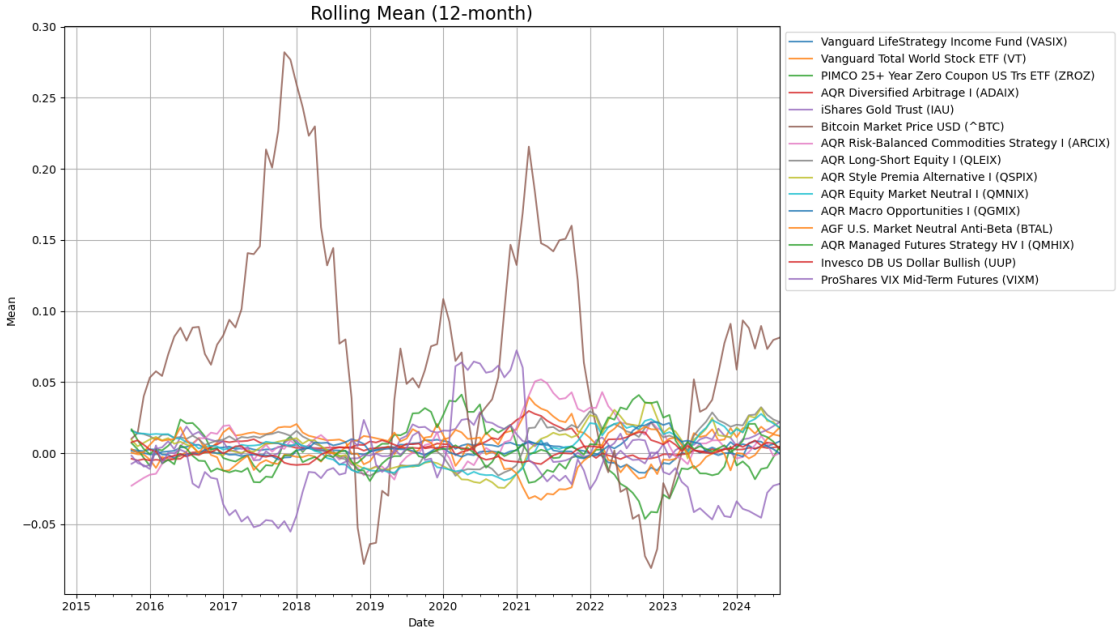
1.0.7 Time Series

[8]:
```
# Descriptive Plots
# Plot Time Series
plt.figure(figsize=(14, 8))

df.plot(ax=plt.gca(), legend=False, alpha=0.8)
plt.title('Time Series of Fund Returns', fontsize=16)
plt.xlabel('Date')
plt.ylabel('Returns')
```

```
plt.grid(True)
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.tight_layout()
plt.show()
```



[9]:
```
# Descriptive Plots
# Plot Time Series - Rolling Mean
rolling_mean = df.rolling(window=12).mean()

plt.figure(figsize=(14, 8))
rolling_mean.plot(ax=plt.gca(), legend=False, alpha=0.8)
plt.title('Rolling Mean (12-month)', fontsize=16)
plt.xlabel('Date')
plt.ylabel('Mean')
plt.grid(True)
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.tight_layout()
plt.show()
```



1.0.8 Correlation Matrix

[10]:
```
# Calculate the correlation matrix between the different funds
correlation_matrix = df.corr()
correlation_matrix
```

[10]:
```
                                                Vanguard LifeStrategy Income
Fund (VASIX)  \
Vanguard LifeStrategy Income Fund (VASIX)
1.000000
Vanguard Total World Stock ETF (VT)
0.797772
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)
0.619699
AQR Diversified Arbitrage I (ADAIX)
0.393475
iShares Gold Trust (IAU)
0.367826
Bitcoin Market Price USD (^BTC)
0.287900
AQR Risk-Balanced Commodities Strategy I (ARCIX)
0.247432
AQR Long-Short Equity I (QLEIX)
0.219485
AQR Style Premia Alternative I (QSPIX)
-0.156694
```

```
AQR Equity Market Neutral I (QMNIX)
-0.231995
AQR Macro Opportunities I (QGMIX)
-0.353287
AGF U.S. Market Neutral Anti-Beta (BTAL)
-0.396115
AQR Managed Futures Strategy HV I (QMHIX)
-0.431737
Invesco DB US Dollar Bullish (UUP)
-0.511921
ProShares VIX Mid-Term Futures (VIXM)
-0.485680

                                                    Vanguard Total World Stock ETF
(VT)  \
Vanguard LifeStrategy Income Fund (VASIX)
0.797772
Vanguard Total World Stock ETF (VT)
1.000000
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)
0.109775
AQR Diversified Arbitrage I (ADAIX)
0.514891
iShares Gold Trust (IAU)
0.136955
Bitcoin Market Price USD (^BTC)
0.324214
AQR Risk-Balanced Commodities Strategy I (ARCIX)
0.477805
AQR Long-Short Equity I (QLEIX)
0.460751
AQR Style Premia Alternative I (QSPIX)
0.007711
AQR Equity Market Neutral I (QMNIX)
-0.139991
AQR Macro Opportunities I (QGMIX)
-0.080182
AGF U.S. Market Neutral Anti-Beta (BTAL)
-0.640102
AQR Managed Futures Strategy HV I (QMHIX)
-0.343223
Invesco DB US Dollar Bullish (UUP)
-0.495324
ProShares VIX Mid-Term Futures (VIXM)
-0.718278

                                                    PIMCO 25+ Year Zero Coupon US
```

```
Trs ETF (ZROZ)  \
Vanguard LifeStrategy Income Fund (VASIX)
0.619699
Vanguard Total World Stock ETF (VT)
0.109775
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)
1.000000
AQR Diversified Arbitrage I (ADAIX)
-0.008409
iShares Gold Trust (IAU)
0.419812
Bitcoin Market Price USD (^BTC)
0.083288
AQR Risk-Balanced Commodities Strategy I (ARCIX)
-0.150134
AQR Long-Short Equity I (QLEIX)
-0.202543
AQR Style Premia Alternative I (QSPIX)
-0.287704
AQR Equity Market Neutral I (QMNIX)
-0.238885
AQR Macro Opportunities I (QGMIX)
-0.322309
AGF U.S. Market Neutral Anti-Beta (BTAL)
0.119725
AQR Managed Futures Strategy HV I (QMHIX)
-0.114056
Invesco DB US Dollar Bullish (UUP)
-0.225970
ProShares VIX Mid-Term Futures (VIXM)
0.064462

                                                    AQR Diversified Arbitrage I
(ADAIX)  \
Vanguard LifeStrategy Income Fund (VASIX)
0.393475
Vanguard Total World Stock ETF (VT)
0.514891
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)
-0.008409
AQR Diversified Arbitrage I (ADAIX)
1.000000
iShares Gold Trust (IAU)
0.036317
Bitcoin Market Price USD (^BTC)
0.277072
AQR Risk-Balanced Commodities Strategy I (ARCIX)
```

```
0.439992
AQR Long-Short Equity I (QLEIX)
0.085852
AQR Style Premia Alternative I (QSPIX)
-0.100675
AQR Equity Market Neutral I (QMNIX)
-0.231412
AQR Macro Opportunities I (QGMIX)
0.115521
AGF U.S. Market Neutral Anti-Beta (BTAL)
-0.489470
AQR Managed Futures Strategy HV I (QMHIX)
-0.214968
Invesco DB US Dollar Bullish (UUP)
-0.274885
ProShares VIX Mid-Term Futures (VIXM)
-0.345754
```

	iShares Gold Trust (IAU)
Vanguard LifeStrategy Income Fund (VASIX)	0.367826
Vanguard Total World Stock ETF (VT)	0.136955
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)	0.419812
AQR Diversified Arbitrage I (ADAIX)	0.036317
iShares Gold Trust (IAU)	1.000000
Bitcoin Market Price USD (^BTC)	0.089561
AQR Risk-Balanced Commodities Strategy I (ARCIX)	0.380276
AQR Long-Short Equity I (QLEIX)	0.033452
AQR Style Premia Alternative I (QSPIX)	-0.152538
AQR Equity Market Neutral I (QMNIX)	-0.013239
AQR Macro Opportunities I (QGMIX)	-0.158983
AGF U.S. Market Neutral Anti-Beta (BTAL)	0.165409
AQR Managed Futures Strategy HV I (QMHIX)	0.058928
Invesco DB US Dollar Bullish (UUP)	-0.458180
ProShares VIX Mid-Term Futures (VIXM)	0.099998

```
                                                Bitcoin Market Price USD
(^BTC)  \
Vanguard LifeStrategy Income Fund (VASIX)
0.287900
Vanguard Total World Stock ETF (VT)
0.324214
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)
0.083288
AQR Diversified Arbitrage I (ADAIX)
0.277072
iShares Gold Trust (IAU)
0.089561
```

```
Bitcoin Market Price USD (^BTC)
1.000000
AQR Risk-Balanced Commodities Strategy I (ARCIX)
0.168327
AQR Long-Short Equity I (QLEIX)
0.097437
AQR Style Premia Alternative I (QSPIX)
-0.037460
AQR Equity Market Neutral I (QMNIX)
-0.116087
AQR Macro Opportunities I (QGMIX)
-0.062044
AGF U.S. Market Neutral Anti-Beta (BTAL)
-0.223792
AQR Managed Futures Strategy HV I (QMHIX)
-0.047346
Invesco DB US Dollar Bullish (UUP)
-0.153065
ProShares VIX Mid-Term Futures (VIXM)
-0.217431

                                                    AQR Risk-Balanced Commodities
Strategy I (ARCIX)  \
Vanguard LifeStrategy Income Fund (VASIX)
0.247432
Vanguard Total World Stock ETF (VT)
0.477805
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)
-0.150134
AQR Diversified Arbitrage I (ADAIX)
0.439992
iShares Gold Trust (IAU)
0.380276
Bitcoin Market Price USD (^BTC)
0.168327
AQR Risk-Balanced Commodities Strategy I (ARCIX)
1.000000
AQR Long-Short Equity I (QLEIX)
0.358372
AQR Style Premia Alternative I (QSPIX)
0.140950
AQR Equity Market Neutral I (QMNIX)
0.086284
AQR Macro Opportunities I (QGMIX)
0.107711
AGF U.S. Market Neutral Anti-Beta (BTAL)
-0.319044
```

```
AQR Managed Futures Strategy HV I (QMHIX)
-0.040040
Invesco DB US Dollar Bullish (UUP)
-0.468701
ProShares VIX Mid-Term Futures (VIXM)
-0.341527

                                                    AQR Long-Short Equity I
(QLEIX)  \
Vanguard LifeStrategy Income Fund (VASIX)
0.219485
Vanguard Total World Stock ETF (VT)
0.460751
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)
-0.202543
AQR Diversified Arbitrage I (ADAIX)
0.085852
iShares Gold Trust (IAU)
0.033452
Bitcoin Market Price USD (^BTC)
0.097437
AQR Risk-Balanced Commodities Strategy I (ARCIX)
0.358372
AQR Long-Short Equity I (QLEIX)
1.000000
AQR Style Premia Alternative I (QSPIX)
0.729026
AQR Equity Market Neutral I (QMNIX)
0.798438
AQR Macro Opportunities I (QGMIX)
0.070964
AGF U.S. Market Neutral Anti-Beta (BTAL)
-0.181864
AQR Managed Futures Strategy HV I (QMHIX)
0.031753
Invesco DB US Dollar Bullish (UUP)
-0.146403
ProShares VIX Mid-Term Futures (VIXM)
-0.497014

                                                    AQR Style Premia Alternative I
(QSPIX)  \
Vanguard LifeStrategy Income Fund (VASIX)
-0.156694
Vanguard Total World Stock ETF (VT)
0.007711
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)
```

```
-0.287704
AQR Diversified Arbitrage I (ADAIX)
-0.100675
iShares Gold Trust (IAU)
-0.152538
Bitcoin Market Price USD (^BTC)
-0.037460
AQR Risk-Balanced Commodities Strategy I (ARCIX)
0.140950
AQR Long-Short Equity I (QLEIX)
0.729026
AQR Style Premia Alternative I (QSPIX)
1.000000
AQR Equity Market Neutral I (QMNIX)
0.806121
AQR Macro Opportunities I (QGMIX)
0.215109
AGF U.S. Market Neutral Anti-Beta (BTAL)
0.158212
AQR Managed Futures Strategy HV I (QMHIX)
0.189893
Invesco DB US Dollar Bullish (UUP)
0.091452
ProShares VIX Mid-Term Futures (VIXM)
-0.157005

                                                AQR Equity Market Neutral I
(QMNIX)  \
Vanguard LifeStrategy Income Fund (VASIX)
-0.231995
Vanguard Total World Stock ETF (VT)
-0.139991
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)
-0.238885
AQR Diversified Arbitrage I (ADAIX)
-0.231412
iShares Gold Trust (IAU)
-0.013239
Bitcoin Market Price USD (^BTC)
-0.116087
AQR Risk-Balanced Commodities Strategy I (ARCIX)
0.086284
AQR Long-Short Equity I (QLEIX)
0.798438
AQR Style Premia Alternative I (QSPIX)
0.806121
AQR Equity Market Neutral I (QMNIX)
```

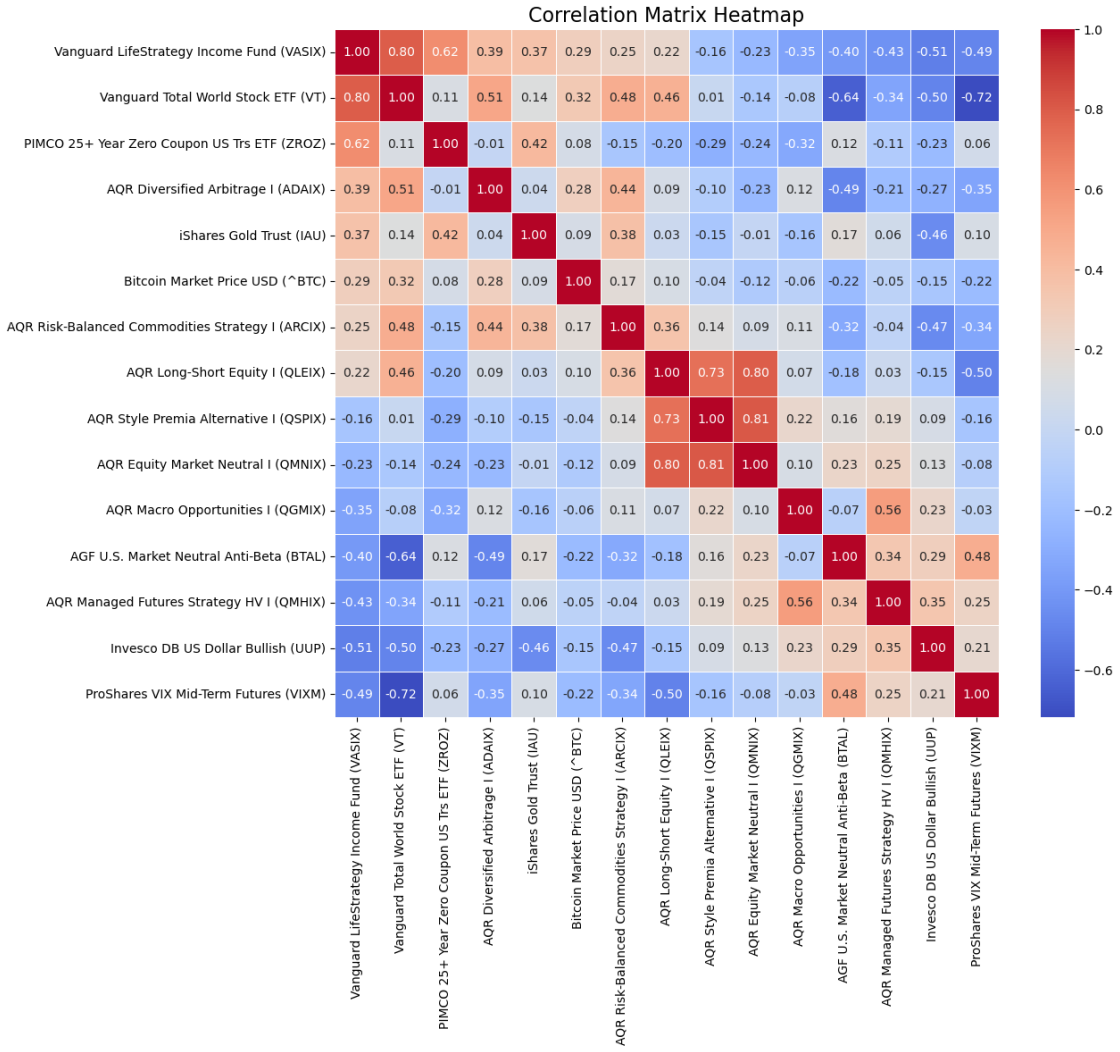
```
1.000000
AQR Macro Opportunities I (QGMIX)
0.097228
AGF U.S. Market Neutral Anti-Beta (BTAL)
0.231500
AQR Managed Futures Strategy HV I (QMHIX)
0.251190
Invesco DB US Dollar Bullish (UUP)
0.129102
ProShares VIX Mid-Term Futures (VIXM)
-0.082435

                                                    AQR Macro Opportunities I
(QGMIX)  \
Vanguard LifeStrategy Income Fund (VASIX)
-0.353287
Vanguard Total World Stock ETF (VT)
-0.080182
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)
-0.322309
AQR Diversified Arbitrage I (ADAIX)
0.115521
iShares Gold Trust (IAU)
-0.158983
Bitcoin Market Price USD (^BTC)
-0.062044
AQR Risk-Balanced Commodities Strategy I (ARCIX)
0.107711
AQR Long-Short Equity I (QLEIX)
0.070964
AQR Style Premia Alternative I (QSPIX)
0.215109
AQR Equity Market Neutral I (QMNIX)
0.097228
AQR Macro Opportunities I (QGMIX)
1.000000
AGF U.S. Market Neutral Anti-Beta (BTAL)
-0.067689
AQR Managed Futures Strategy HV I (QMHIX)
0.556294
Invesco DB US Dollar Bullish (UUP)
0.226025
ProShares VIX Mid-Term Futures (VIXM)
-0.029806

                                                    AGF U.S. Market Neutral Anti-
Beta (BTAL)  \
```

```
Vanguard LifeStrategy Income Fund (VASIX)
-0.396115
Vanguard Total World Stock ETF (VT)
-0.640102
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)
0.119725
AQR Diversified Arbitrage I (ADAIX)
-0.489470
iShares Gold Trust (IAU)
0.165409
Bitcoin Market Price USD (^BTC)
-0.223792
AQR Risk-Balanced Commodities Strategy I (ARCIX)
-0.319044
AQR Long-Short Equity I (QLEIX)
-0.181864
AQR Style Premia Alternative I (QSPIX)
0.158212
AQR Equity Market Neutral I (QMNIX)
0.231500
AQR Macro Opportunities I (QGMIX)
-0.067689
AGF U.S. Market Neutral Anti-Beta (BTAL)
1.000000
AQR Managed Futures Strategy HV I (QMHIX)
0.341463
Invesco DB US Dollar Bullish (UUP)
0.288372
ProShares VIX Mid-Term Futures (VIXM)
0.481303

                                                    AQR Managed Futures Strategy
HV I (QMHIX)  \
Vanguard LifeStrategy Income Fund (VASIX)
-0.431737
Vanguard Total World Stock ETF (VT)
-0.343223
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)
-0.114056
AQR Diversified Arbitrage I (ADAIX)
-0.214968
iShares Gold Trust (IAU)
0.058928
Bitcoin Market Price USD (^BTC)
-0.047346
AQR Risk-Balanced Commodities Strategy I (ARCIX)
-0.040040
```

```
AQR Long-Short Equity I (QLEIX)
0.031753
AQR Style Premia Alternative I (QSPIX)
0.189893
AQR Equity Market Neutral I (QMNIX)
0.251190
AQR Macro Opportunities I (QGMIX)
0.556294
AGF U.S. Market Neutral Anti-Beta (BTAL)
0.341463
AQR Managed Futures Strategy HV I (QMHIX)
1.000000
Invesco DB US Dollar Bullish (UUP)
0.347456
ProShares VIX Mid-Term Futures (VIXM)
0.254549

                                                    Invesco DB US Dollar Bullish
(UUP)  \
Vanguard LifeStrategy Income Fund (VASIX)
-0.511921
Vanguard Total World Stock ETF (VT)
-0.495324
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)
-0.225970
AQR Diversified Arbitrage I (ADAIX)
-0.274885
iShares Gold Trust (IAU)
-0.458180
Bitcoin Market Price USD (^BTC)
-0.153065
AQR Risk-Balanced Commodities Strategy I (ARCIX)
-0.468701
AQR Long-Short Equity I (QLEIX)
-0.146403
AQR Style Premia Alternative I (QSPIX)
0.091452
AQR Equity Market Neutral I (QMNIX)
0.129102
AQR Macro Opportunities I (QGMIX)
0.226025
AGF U.S. Market Neutral Anti-Beta (BTAL)
0.288372
AQR Managed Futures Strategy HV I (QMHIX)
0.347456
Invesco DB US Dollar Bullish (UUP)
1.000000
```

```
ProShares VIX Mid-Term Futures (VIXM)
0.206555

                                                    ProShares VIX Mid-Term Futures
(VIXM)
Vanguard LifeStrategy Income Fund (VASIX)
-0.485680
Vanguard Total World Stock ETF (VT)
-0.718278
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)
0.064462
AQR Diversified Arbitrage I (ADAIX)
-0.345754
iShares Gold Trust (IAU)
0.099998
Bitcoin Market Price USD (^BTC)
-0.217431
AQR Risk-Balanced Commodities Strategy I (ARCIX)
-0.341527
AQR Long-Short Equity I (QLEIX)
-0.497014
AQR Style Premia Alternative I (QSPIX)
-0.157005
AQR Equity Market Neutral I (QMNIX)
-0.082435
AQR Macro Opportunities I (QGMIX)
-0.029806
AGF U.S. Market Neutral Anti-Beta (BTAL)
0.481303
AQR Managed Futures Strategy HV I (QMHIX)
0.254549
Invesco DB US Dollar Bullish (UUP)
0.206555
ProShares VIX Mid-Term Futures (VIXM)
1.000000
```

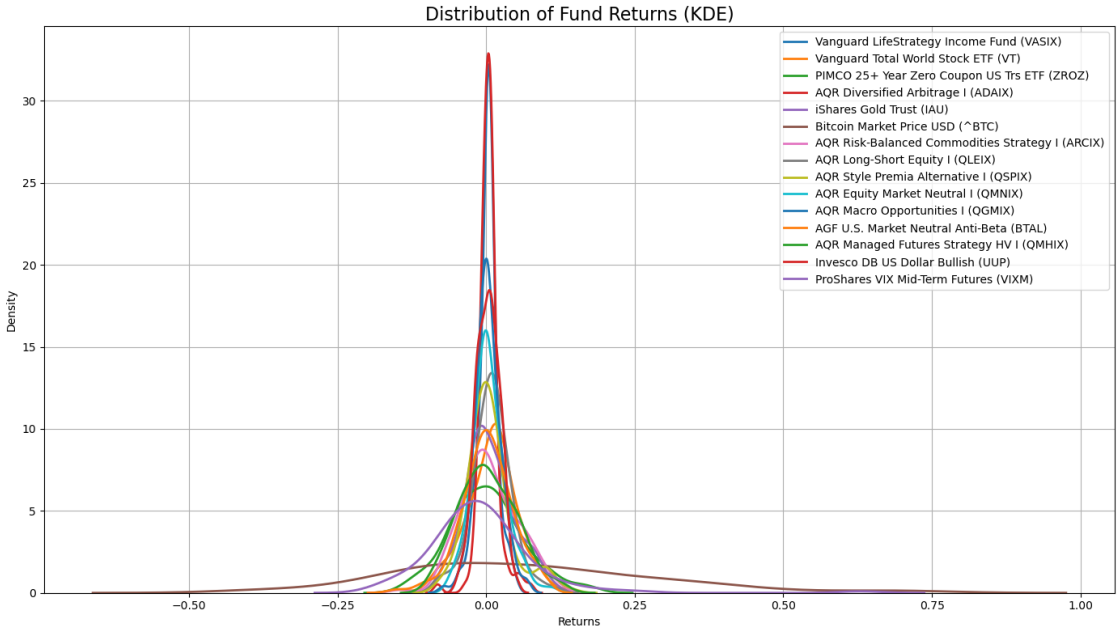
[11]:
```
# Visualize the correlation matrix using a heatmap
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f',␣
 ↪linewidths=0.5)
plt.title('Correlation Matrix Heatmap', fontsize=16)
plt.show()
```



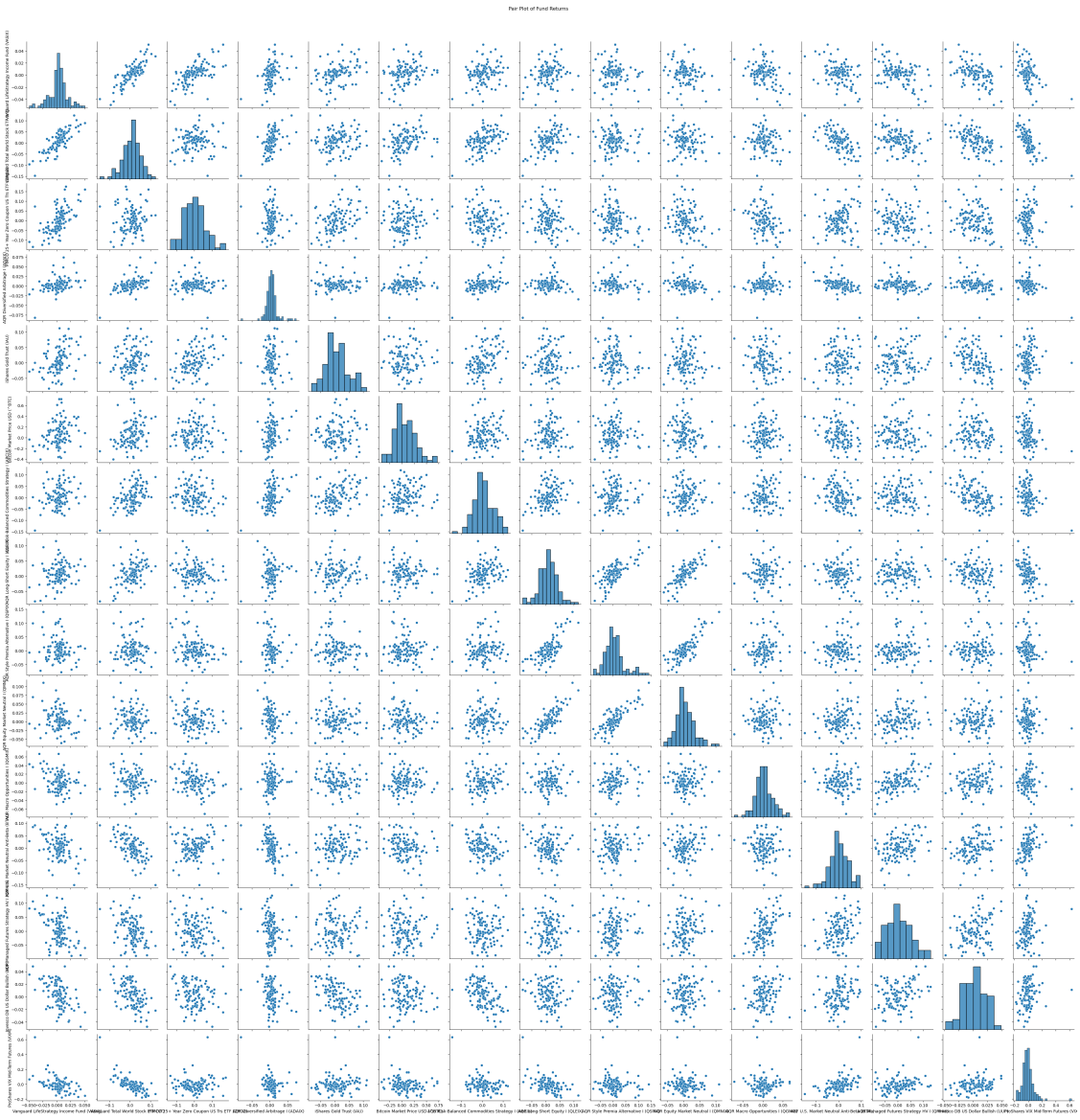
1.0.9 Kernel Density Estimate

[12]:
```
# Plot the# Plot the Kernel Density Estimate for each fund
plt.figure(figsize=(14, 8))

for column in df.columns:
    sns.kdeplot(df[column], label=column, linewidth=2)

plt.title('Distribution of Fund Returns (KDE)', fontsize=16)
plt.xlabel('Returns')
plt.ylabel('Density')
plt.grid(True)
plt.legend(loc='upper right', bbox_to_anchor=(1, 1))
plt.tight_layout()
```

```
plt.show()
```



[13]:
```
# Create a pair plot for the dataset to visualize relationships between funds
import seaborn as sns
pairplot_df = df
sns.pairplot(pairplot_df)
plt.suptitle("Pair Plot of Fund Returns", y=1.02)
plt.show()
```



From ChatGPT

1.0.10 How to Read Pair Plot

Key Elements of a Pair Plot: Scatter Plots (Off-Diagonal): Each scatter plot shows the relationship between two different funds. Each point represents an observation (a time point) with the return of one fund on the x-axis and the return of another fund on the y-axis.

Linear Relationships: A clear upward or downward trend indicates a positive or negative correlation, respectively, between the returns of two funds. No Pattern: If the points are scattered without any clear direction, this implies little or no correlation between the two funds. Clusters: Visible clusters or groupings of points may indicate different regimes or types of behavior in the data (e.g., periods of high/low returns). Histograms (Diagonal): The diagonal plots show the dis-

tribution of returns for each fund individually. These are histograms (or sometimes density plots), allowing you to assess the shape of the distribution.

Normal Distribution: If the histogram is bell-shaped, the returns of the fund are approximately normally distributed. Skewness: If the histogram leans to the left or right, the returns are skewed. Spread: The width of the histogram indicates the variability or volatility of the returns. A wide histogram suggests higher variability (volatility) in returns.

1.0.11 Interpreting Scatter Plots

Positive Correlation: If the points in the scatter plot form an upward-sloping line, it indicates that when the return of one fund increases, the return of the other tends to increase as well. The stronger the correlation, the tighter the points will cluster along the line.

Example: If two funds are positively correlated, investing in both may not diversify risk because their performance moves together.

Negative Correlation: A downward-sloping line means the two funds are negatively correlated, where one increases while the other decreases. This can be a sign that the two funds might hedge each other.

Example: If one fund tends to gain when another loses, holding both may reduce portfolio risk.

No Correlation: A random scatter of points with no discernible pattern suggests no relationship between the two funds.

1.0.12 Interpreting Histograms (Diagonal):

Symmetry: Symmetrical histograms show that the returns are balanced around the mean (close to normal distribution). #### Skewness: **Right Skewed (Positive Skew):** Most of the returns are concentrated on the left side, with a long tail on the right. This indicates occasional large gains. **Left Skewed (Negative Skew):** Most returns are concentrated on the right side, with a long tail on the left, indicating occasional large losses. **Kurtosis (Fat Tails):** If the histogram has high peaks and fat tails, it suggests the presence of extreme values, meaning the fund might experience rare but large fluctuations in returns.

1.0.13 Models

Risk Parity

[14]:
```
# Load Opportunity Dataset
data = pd.read_csv('Opportunity_Set.csv')      # Load Dataset as Dataframe
data.head()
 ↪       # Diaplay 1st 5 in Dataframe
```

[14]:

	Date	Vanguard LifeStrategy Income Fund (VASIX)	\
0	11/30/2014	0.0094	
1	12/31/2014	-0.0005	
2	1/31/2015	0.0141	
3	2/28/2015	0.0033	
4	3/31/2015	0.0018	

```
   Vanguard Total World Stock ETF (VT)  \
0                               0.0126
1                              -0.0199
2                              -0.0163
3                               0.0595
4                              -0.0121

   PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)  \
0                                        0.0414
1                                        0.0612
2                                        0.1600
3                                       -0.1007
4                                        0.0117

   AQR Diversified Arbitrage I (ADAIX)  iShares Gold Trust (IAU)  \
0                              -0.0066                   -0.0053
1                              -0.0117                    0.0133
2                              -0.0059                    0.0865
3                               0.0069                   -0.0579
4                               0.0000                   -0.0222

   Bitcoin Market Price USD (^BTC)  \
0                           0.0969
1                          -0.1777
2                          -0.2677
3                           0.1062
4                          -0.0150

   AQR Risk-Balanced Commodities Strategy I (ARCIX)  \
0                                           -0.0726
1                                           -0.0412
2                                           -0.0287
3                                            0.0044
4                                           -0.0573

   AQR Long-Short Equity I (QLEIX)  AQR Style Premia Alternative I (QSPIX)  \
0                           0.0248                                  0.0412
1                           0.0140                                  0.0002
2                           0.0156                                 -0.0112
3                           0.0236                                 -0.0390
4                          -0.0027                                  0.0256

   AQR Equity Market Neutral I (QMNIX)  AQR Macro Opportunities I (QGMIX)  \
0                               0.0257                                0.0154
1                               0.0195                                0.0039
2                               0.0290                               -0.0070
3                              -0.0078                                0.0091
```

```
4                                0.0049                                0.0261

   AGF U.S. Market Neutral Anti-Beta (BTAL)  \
0                                    0.0235
1                                    0.0294
2                                    0.0320
3                                   -0.0568
4                                    0.0000

   AQR Managed Futures Strategy HV I (QMHIX)  \
0                                     0.1159
1                                     0.0461
2                                     0.0721
3                                    -0.0108
4                                     0.0655

   Invesco DB US Dollar Bullish (UUP)  ProShares VIX Mid-Term Futures (VIXM)
0                              0.0165                                -0.0298
1                              0.0213                                 0.0553
2                              0.0484                                 0.0762
3                              0.0028                                -0.1145
4                              0.0278                                 0.0033
```

[15]:
```python
# Extract the returns data (excluding the Date column)
returns = data.iloc[:, 1:]

# Calculate the covariance matrix of asset returns
cov_matrix = returns.cov()

# Number of assets
n_assets = cov_matrix.shape[0]

# Define the variables for the optimization (portfolio weights)
weights = cp.Variable(n_assets)

# Define the objective (minimize portfolio variance)
portfolio_variance = cp.quad_form(weights, cov_matrix.values)

# Constraints (weights sum to 1 and are non-negative)
constraints = [cp.sum(weights) == 1, weights >= 0]
# the constraints don't seem to work with negative optimal weight below...???

# Optimization problem (minimize variance)
problem = cp.Problem(cp.Minimize(portfolio_variance), constraints)
problem.solve()

# Optimal portfolio weights
```

```
optimal_weights = weights.value

# Compute Marginal Risk Contribution (MRC)
mrc = 2 * np.dot(cov_matrix.values, optimal_weights)

# Creating a dataframe to display the results
mrc_df = pd.DataFrame({
    'Asset': returns.columns,
    'Optimal Weights': optimal_weights,
    'Marginal Risk Contribution': mrc
})

# Display the results
mrc_df
```

[15]:

	Asset	Optimal Weights	Marginal Risk Contribution
0	Vanguard LifeStrategy Income Fund (VASIX)	3.932301e-01	0.000061
1	Vanguard Total World Stock ETF (VT)	-2.099375e-18	0.000079
2	PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)	-5.248749e-18	0.000239
3	AQR Diversified Arbitrage I (ADAIX)	1.133455e-01	0.000061
4	iShares Gold Trust (IAU)	-2.448059e-18	0.000098
5	Bitcoin Market Price USD (^BTC)	1.844191e-18	0.000238
6	AQR Risk-Balanced Commodities Strategy I (ARCIX)	2.807962e-02	0.000061
7	AQR Long-Short Equity I (QLEIX)	-5.006550e-19	0.000089
8	AQR Style Premia Alternative I (QSPIX)	4.818605e-19	0.000098
9	AQR Equity Market Neutral I (QMNIX)	7.008853e-02	0.000061
10	AQR Macro Opportunities I (QGMIX)	1.092522e-01	0.000061
11	AGF U.S. Market Neutral Anti-Beta (BTAL)	3.719750e-02	0.000061
12	AQR Managed Futures Strategy HV I (QMHIX)	-9.423550e-19	0.000131
13	Invesco DB US Dollar Bullish (UUP)	2.149796e-01	0.000061
14	ProShares VIX Mid-Term Futures (VIXM)	3.382701e-02	

```
14                    0.000061
```

[16]:
```
# Assuming equal weights for simplicity in this example, but you can replace␣
↪with any other weighting strategy
n_assets = len(returns.columns)
equal_weights = np.array([1/n_assets] * n_assets)

# Calculate the portfolio variance
portfolio_variance_equal_weight = equal_weights.T @ cov_matrix @ equal_weights

# Portfolio variance result
portfolio_variance_equal_weight
```

[16]:
```
0.0003504452730310492
```

[17]:
```
# Calculate the mean returns (expected returns) of each asset
expected_returns = returns.mean()

# Calculate the portfolio's expected return using equal weights
portfolio_expected_return_equal = np.dot(equal_weights, expected_returns)

# Portfolio expected return result
portfolio_expected_return_equal
```

[17]:
```
0.007898983050847456
```

[18]:
```
# Calculate the mean returns (expected returns) of each asset
expected_returns = returns.mean()

# Calculate the portfolio's expected return using optimal weights
portfolio_expected_return_optimal = np.dot(optimal_weights, expected_returns)

# Portfolio optimal return result
portfolio_expected_return_optimal
```

[18]:
```
0.0028506210338368025
```

[19]:
```
# Assume risk-free rate is 0 for simplicity, you can adjust as necessary
risk_free_rate = 0.0

# Portfolio variance and standard deviation
portfolio_variance_optimal = np.dot(optimal_weights.T, np.dot(cov_matrix,␣
↪optimal_weights))
portfolio_std = np.sqrt(portfolio_variance_optimal)

# Sharpe ratio calculation
```

```
portfolio_sharpe_ratio = (portfolio_expected_return_optimal - risk_free_rate) /␣
 ↪portfolio_std

# Display the Sharpe ratio
print(f"Portfolio Sharpe Ratio: {portfolio_sharpe_ratio:.4f}")
```

```
Portfolio Sharpe Ratio: 0.5141
```

[20]:
```
print("Equal Weighted Portfolio")
print(f"The Portfolio Variance is {portfolio_variance_equal_weight:.4f}")
print(f"The Expected Return is {portfolio_expected_return_equal:.4f}")
print()
print("Optimal Weighted Portfolio")
print(f"The Portfolio Variance is {portfolio_variance_optimal:.4f}")
print(f"The Expected Return is {portfolio_expected_return_optimal:.4f}")
print(f"The Portfolio Sharpe Ratio is {portfolio_sharpe_ratio:.4f}")
```

```
Equal Weighted Portfolio
The Portfolio Variance is 0.0004
The Expected Return is 0.0079

Optimal Weighted Portfolio
The Portfolio Variance is 0.0000
The Expected Return is 0.0029
The Portfolio Sharpe Ratio is 0.5141
```

The constraints used to prevent a negative optimal weight do not work in this model (Input 3)

Fama-French Factor Model

[21]:
```
# http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html
fama_french_factors = pd.read_csv('F-F_Research_Data_Factors.csv',␣
 ↪parse_dates=['Date'])      # Load Dataset as Dataframe
fama_french_factors.head()                                                  ␣
 ↪              # Diaplay 1st 5 inDataframe
```

[21]:

	Date	Mkt-RF	SMB	HML	RF
0	2014-11-30	2.55	-2.06	-3.10	0.0
1	2014-12-31	-0.06	2.49	2.27	0.0
2	2015-01-31	-3.11	-0.56	-3.59	0.0
3	2015-02-28	6.13	0.63	-1.86	0.0
4	2015-03-31	-1.12	3.04	-0.38	0.0

[22]:
```
df = pd.read_csv('Opportunity_Set.csv', parse_dates=['Date'])      # Load␣
 ↪Dataset as Dataframe
df.head()
```

[22]:

	Date	Vanguard LifeStrategy Income Fund (VASIX)	Vanguard Total World Stock ETF (VT)	PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)	AQR Diversified Arbitrage I (ADAIX)	iShares Gold Trust (IAU)	Bitcoin Market Price USD (^BTC)	AQR Risk-Balanced Commodities Strategy I (ARCIX)	AQR Long-Short Equity I (QLEIX)	AQR Style Premia Alternative I (QSPIX)
0	2014-11-30	0.0094	0.0126	0.0414	-0.0066	-0.0053	0.0969	-0.0726	0.0248	0.0412
1	2014-12-31	-0.0005	-0.0199	0.0612	-0.0117	0.0133	-0.1777	-0.0412	0.0140	0.0002
2	2015-01-31	0.0141	-0.0163	0.1600	-0.0059	0.0865	-0.2677	-0.0287	0.0156	-0.0112
3	2015-02-28	0.0033	0.0595	-0.1007	0.0069	-0.0579	0.1062	0.0044	0.0236	-0.0390
4	2015-03-31	0.0018	-0.0121	0.0117	0.0000	-0.0222	-0.0150	-0.0573		

```
4                          -0.0027                                    0.0256

   AQR Equity Market Neutral I (QMNIX)  AQR Macro Opportunities I (QGMIX)  \
0                               0.0257                             0.0154
1                               0.0195                             0.0039
2                               0.0290                            -0.0070
3                              -0.0078                             0.0091
4                               0.0049                             0.0261

   AGF U.S. Market Neutral Anti-Beta (BTAL)  \
0                                    0.0235
1                                    0.0294
2                                    0.0320
3                                   -0.0568
4                                    0.0000

   AQR Managed Futures Strategy HV I (QMHIX)  \
0                                     0.1159
1                                     0.0461
2                                     0.0721
3                                    -0.0108
4                                     0.0655

   Invesco DB US Dollar Bullish (UUP)  ProShares VIX Mid-Term Futures (VIXM)
0                              0.0165                                -0.0298
1                              0.0213                                 0.0553
2                              0.0484                                 0.0762
3                              0.0028                                -0.1145
4                              0.0278                                 0.0033
```

[23]:
```
import pandas as pd
import statsmodels.api as sm

asset_return = df['Vanguard LifeStrategy Income Fund (VASIX)']  # Replace with␣
 ↪your asset's returns

# Step 3: Prepare the factors and align with asset returns by date
factor_returns = fama_french_factors[['Mkt-RF', 'SMB', 'HML']] # Ensure both␣
 ↪asset returns and factors are aligned by date
factor_returns_RF = fama_french_factors['RF']  # Risk-free rate
excess_asset_returns = asset_return - factor_returns_RF # Calculate excess␣
 ↪returns over risk-free rate

# Add a constant (alpha) to the model
X = sm.add_constant(factor_returns[['Mkt-RF', 'SMB', 'HML']])
y = excess_asset_returns
```

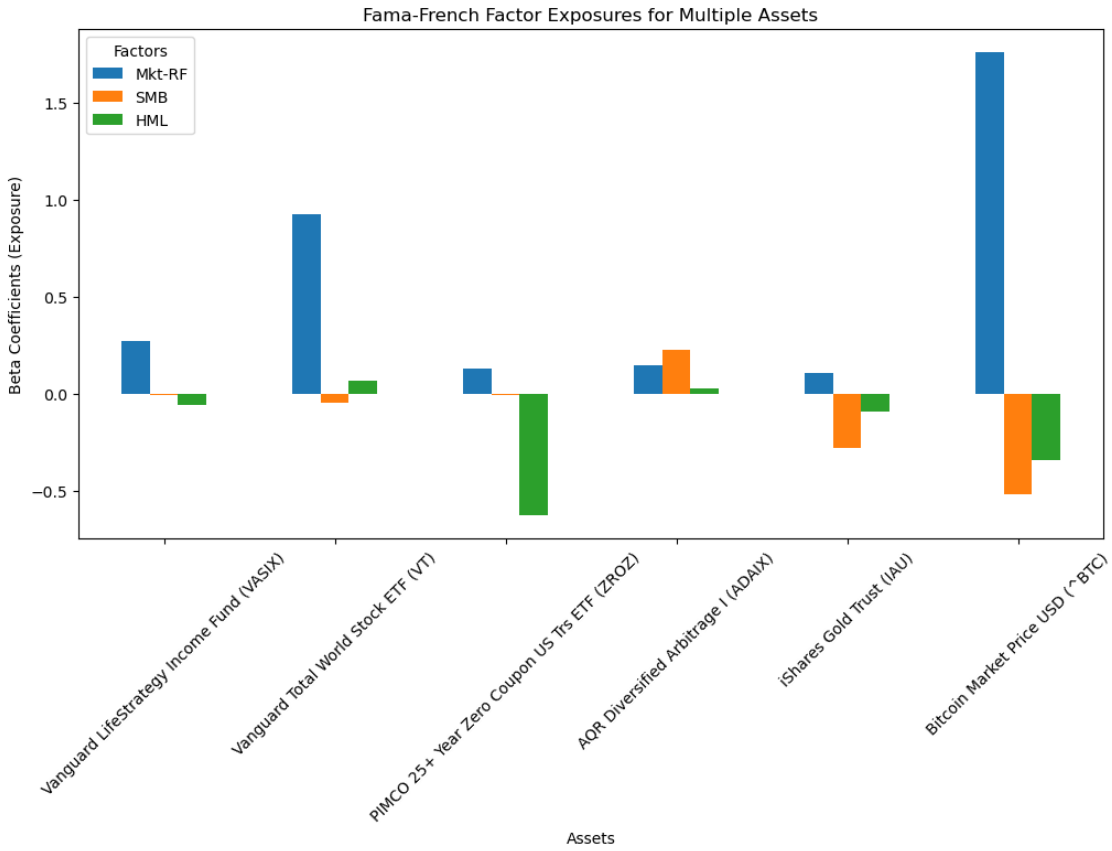
```
# Fit the model
model = sm.OLS(y, X).fit()

# Display the summary of the regression
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.025
Model:                            OLS   Adj. R-squared:                 -0.001
Method:                 Least Squares   F-statistic:                    0.9667
Date:                Sun, 06 Oct 2024   Prob (F-statistic):              0.411
Time:                        12:08:36   Log-Likelihood:                 57.434
No. Observations:                 118   AIC:                            -106.9
Df Residuals:                     114   BIC:                            -95.78
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.1267      0.014     -8.839      0.000      -0.155      -0.098
Mkt-RF         0.0016      0.003      0.518      0.605      -0.005       0.008
SMB            0.0066      0.005      1.260      0.210      -0.004       0.017
HML            0.0020      0.004      0.532      0.596      -0.005       0.009
==============================================================================
Omnibus:                       18.265   Durbin-Watson:                   0.069
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               22.950
Skew:                          -1.080   Prob(JB):                     1.04e-05
Kurtosis:                       2.930   Cond. No.                         4.91
==============================================================================
```

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

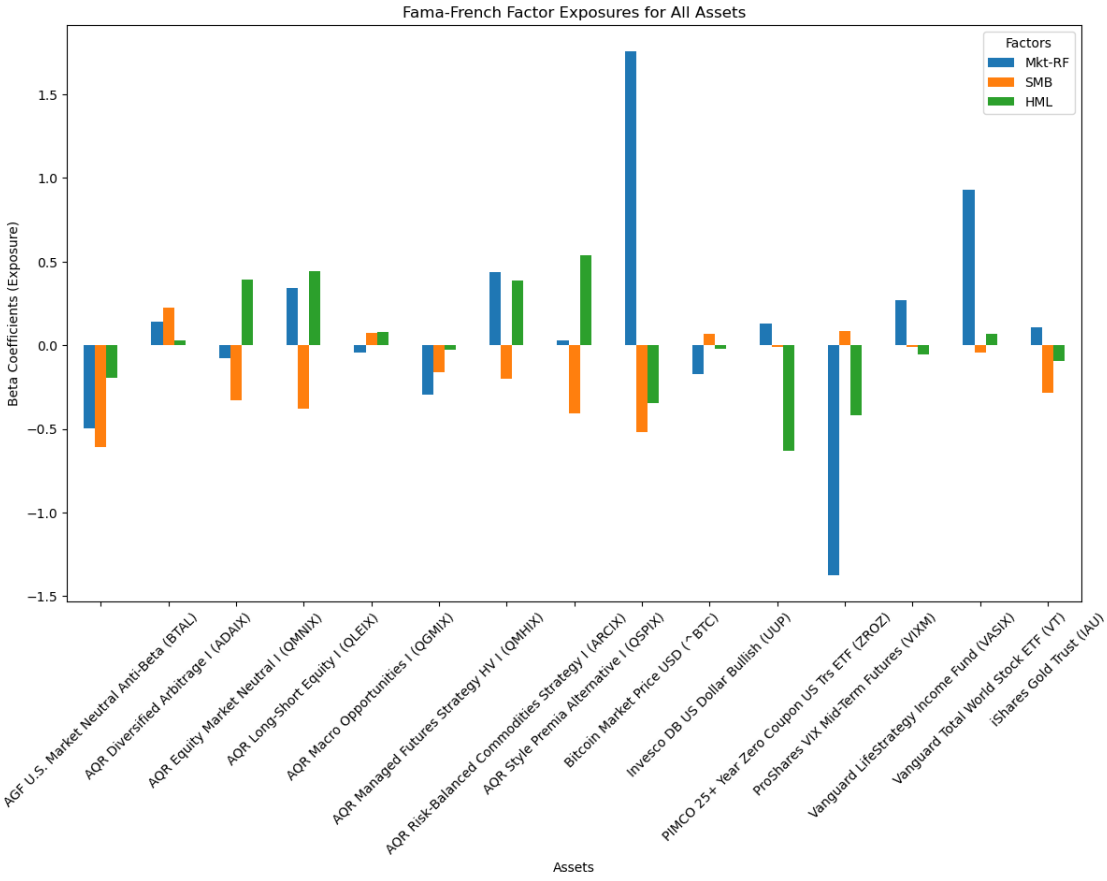
[24]:
```
# Merge the two datasets on the 'Date' column
merged_df = pd.merge(df, fama_french_factors, on='Date')

# Set the index to 'Date' for easier handling
merged_df.set_index('Date', inplace=True)

# Prepare the assets for regression (list of asset columns)
asset_columns = [
    'Vanguard LifeStrategy Income Fund (VASIX)',
    'Vanguard Total World Stock ETF (VT)',
    'PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)',
    'AQR Diversified Arbitrage I (ADAIX)',
```

```
    'iShares Gold Trust (IAU)',
    'Bitcoin Market Price USD (^BTC)'
]

# We need to calculate excess returns for the assets (subtracting the risk-free␣
 ↪rate RF)
for asset in asset_columns:
    merged_df[asset] = merged_df[asset] - (merged_df['RF'] / 100)  # Converting␣
 ↪RF to percentage

# Prepare the factor data for the regression (convert from percentage to␣
 ↪decimal)
factor_columns = ['Mkt-RF', 'SMB', 'HML']
merged_df[factor_columns] = merged_df[factor_columns] / 100

# Perform Fama-French 3-factor regression for each asset
import statsmodels.api as sm

betas = pd.DataFrame()

for asset in asset_columns:
    X = sm.add_constant(merged_df[factor_columns])  # Add constant (alpha)
    y = merged_df[asset]  # Asset's excess returns
    model = sm.OLS(y, X).fit()  # Fit OLS regression

    # Collect the coefficients (betas) for each asset
    betas[asset] = model.params[1:]  # Ignore the intercept (alpha), take only␣
 ↪factor betas

# Rename the index for factors
betas.index = factor_columns
betas

# Now we can visualize the beta exposures for all assets
betas.T.plot(kind='bar', figsize=(12, 6))
plt.title('Fama-French Factor Exposures for Multiple Assets')
plt.xlabel('Assets')
plt.ylabel('Beta Coefficients (Exposure)')
plt.xticks(rotation=45)
plt.legend(title='Factors')

# Display the plot
plt.show()
```



[25]:
```
# Let's extend the analysis to include all assets in the opportunity dataset.

# Extract all asset columns (ignoring the factor columns)
all_asset_columns = merged_df.columns.difference(factor_columns + ['RF']).
 ↪tolist()

# Initialize an empty DataFrame to store the betas for all assets
all_betas = pd.DataFrame()

# Perform Fama-French 3-factor regression for each asset
for asset in all_asset_columns:
    X = sm.add_constant(merged_df[factor_columns])  # Add constant (alpha)
    y = merged_df[asset]  # Asset's excess returns
    model = sm.OLS(y, X).fit()  # Fit OLS regression

    # Collect the coefficients (betas) for each asset
    all_betas[asset] = model.params[1:]  # Ignore the intercept (alpha), take␣
 ↪only factor betas

# Rename the index for factors
```

```
all_betas.index = factor_columns

all_betas

# Visualize the beta exposures for all assets
all_betas.T.plot(kind='bar', figsize=(14, 8))
plt.title('Fama-French Factor Exposures for All Assets')
plt.xlabel('Assets')
plt.ylabel('Beta Coefficients (Exposure)')
plt.xticks(rotation=45)
plt.legend(title='Factors')

# Display the plot
plt.show()
```



Betas (1, 2, 3) : These represent the asset's sensitivity to each factor.

A high 1 (market) means the asset is strongly correlated with market movements. A positive 2 (SMB) means the asset has small-cap exposure. A positive 3 (HML) means the asset has exposure to value stocks.

References

Agresti, Alan, and Maria Kateri. Foundations of Statistics for Data Scientists: With R and Python. CRC Press, Taylor & Francis Group, 2022.

Agresti, Alan, and Maria Kateri. (2022) Appendix B. In Foundations of Statistics for Data Scientists: With R and Python (p. 385-389). CRC Press, Taylor & Francis Group, 2022.

ChatGPT, (2024) GPT-4o version, OpenAI. [Large language model]. https://chatgpt.com/

Opportuinty Dataset - SRL Global Ltd. (2024, September 16). Backtest portfolio asset allocation. Portfolio Visualizer. Retrieved September 16, 2024, from https://www.portfoliovisualizer.com/backtest-Portfolio?s=y&sl=5M8RhWX2CyedGtfIDJRlyP

F-F_Reasearch_Data_Factors.CSV Kenneth French's website. http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html

Project GitHub Repository: https://github.com/littlecl42/AAA500_FinalProject_CL_IL/