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
```

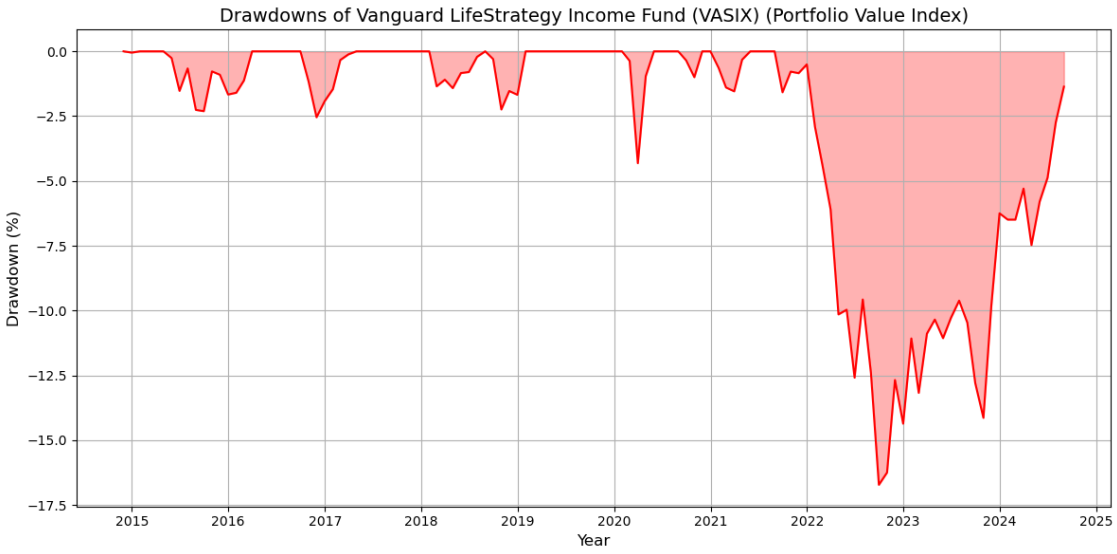
2	0.0484	0.0762
3	0.0028	-0.1145
4	0.0278	0.0033

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()
```

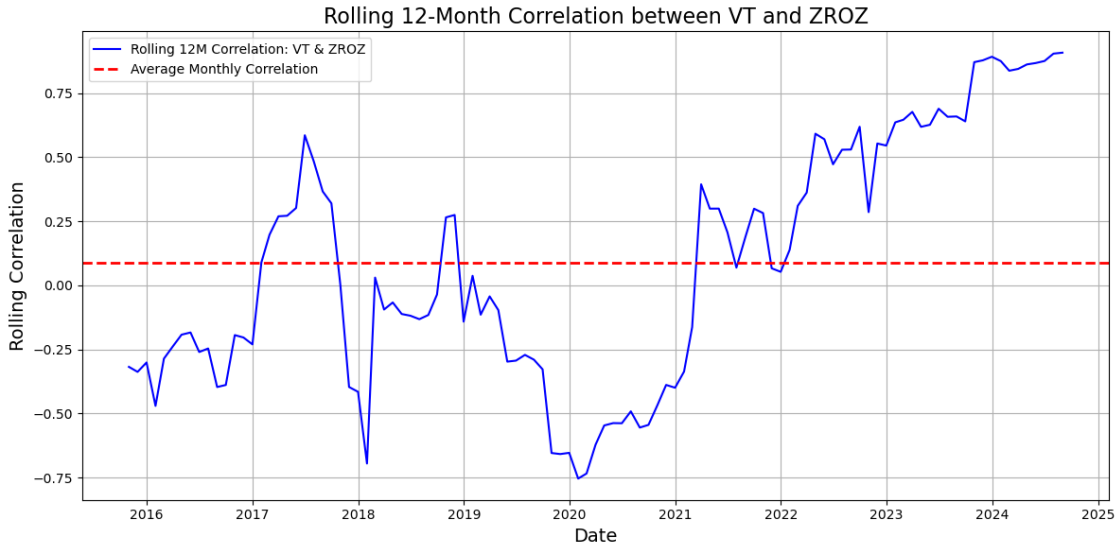


3 Rolling 12-month Correlation Between Stocks and Bonds (Figure 2)

[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))
```

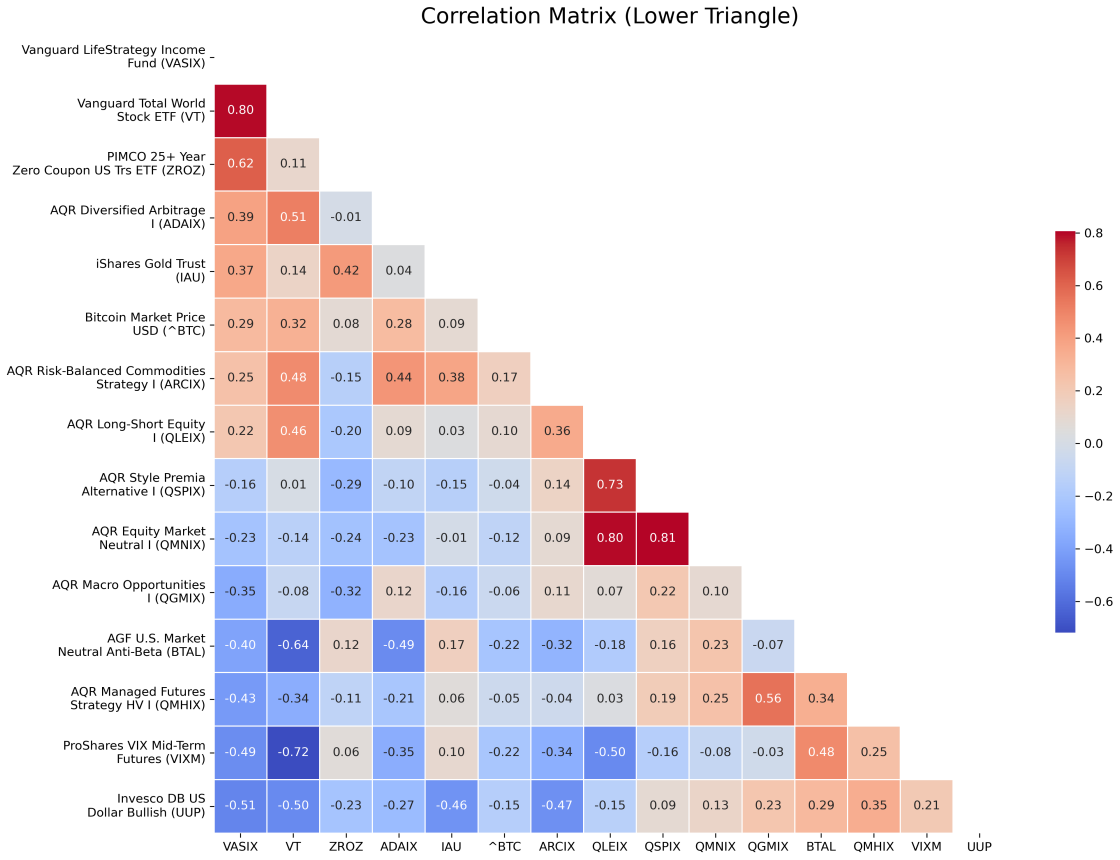
```
    plt.plot(corr_series.index, corr_series, color='blue', label=f'Rolling 12M␣
↪Correlation: {asset1} & {asset2}')

    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)

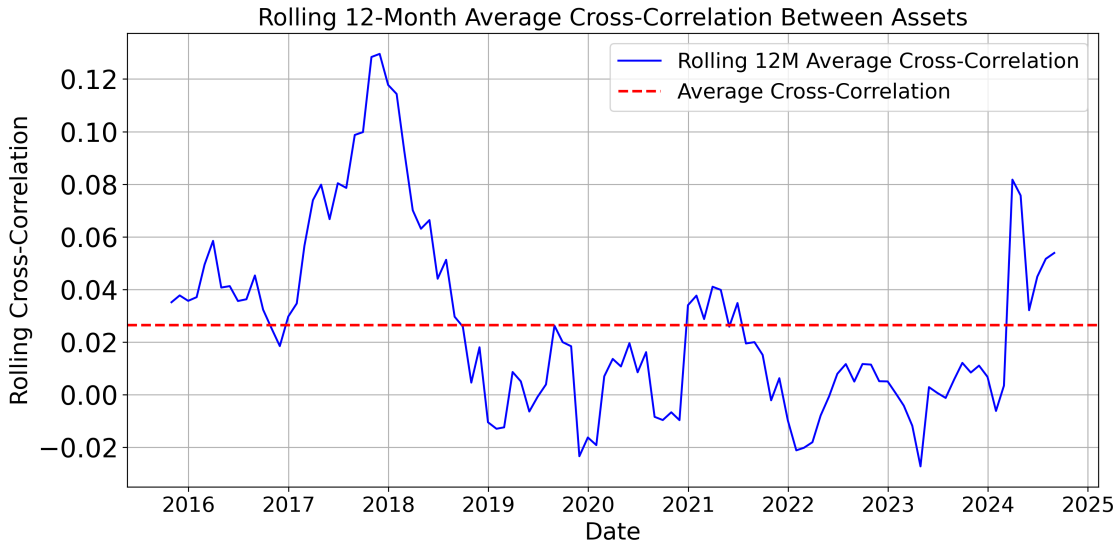
[6]:
```
import os
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from PIL import Image
from IPython.display import display
from scipy import stats
import numpy as np

# ------------------------------ #
#         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']
```

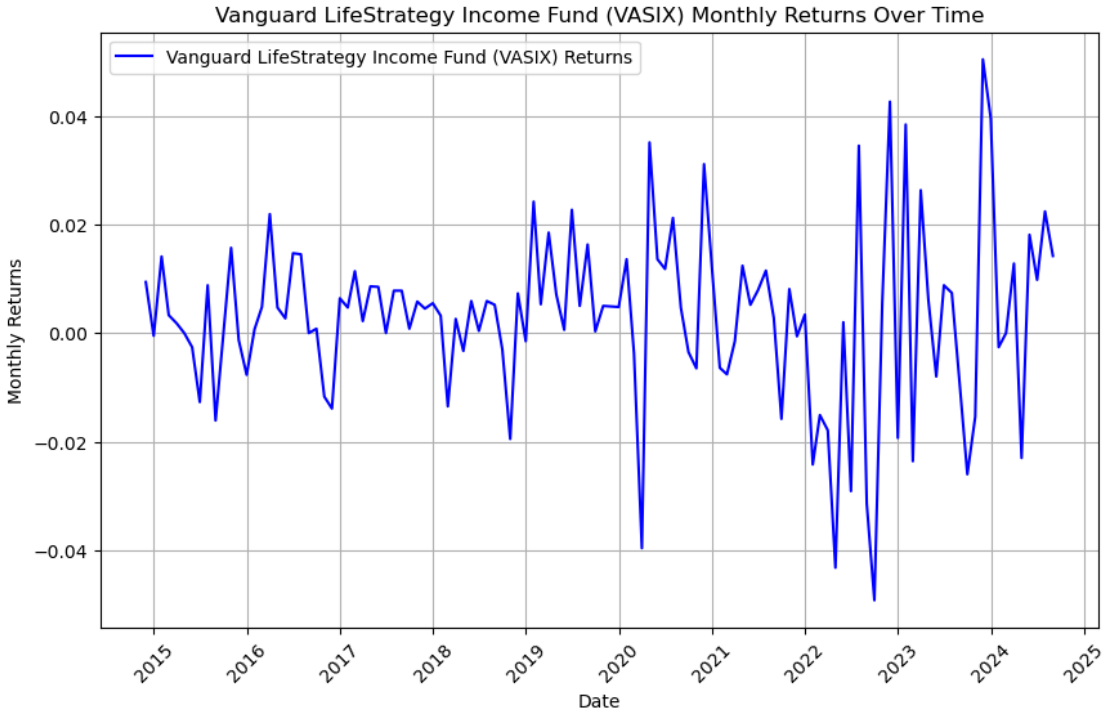
```
# Add Kurtosis and Skewness
summary_stats['Skewness'] = skewness
summary_stats['Kurtosis'] = kurtosis

# 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
```

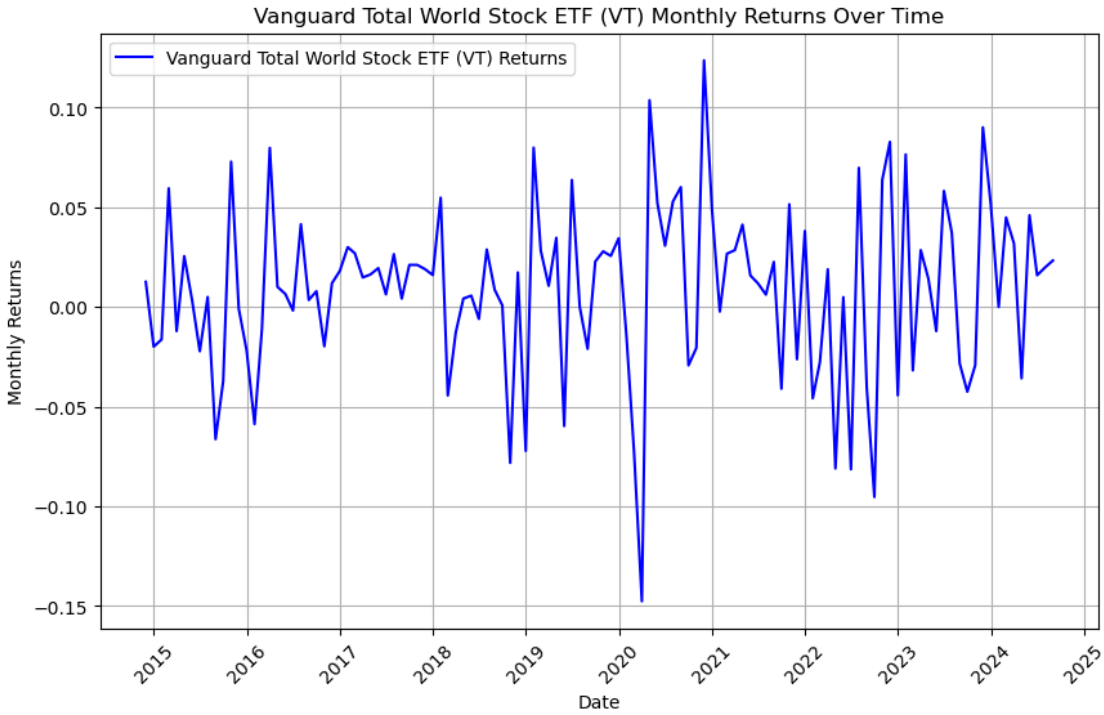
[221]: <pandas.io.formats.style.Styler at 0x30e4a9a30>

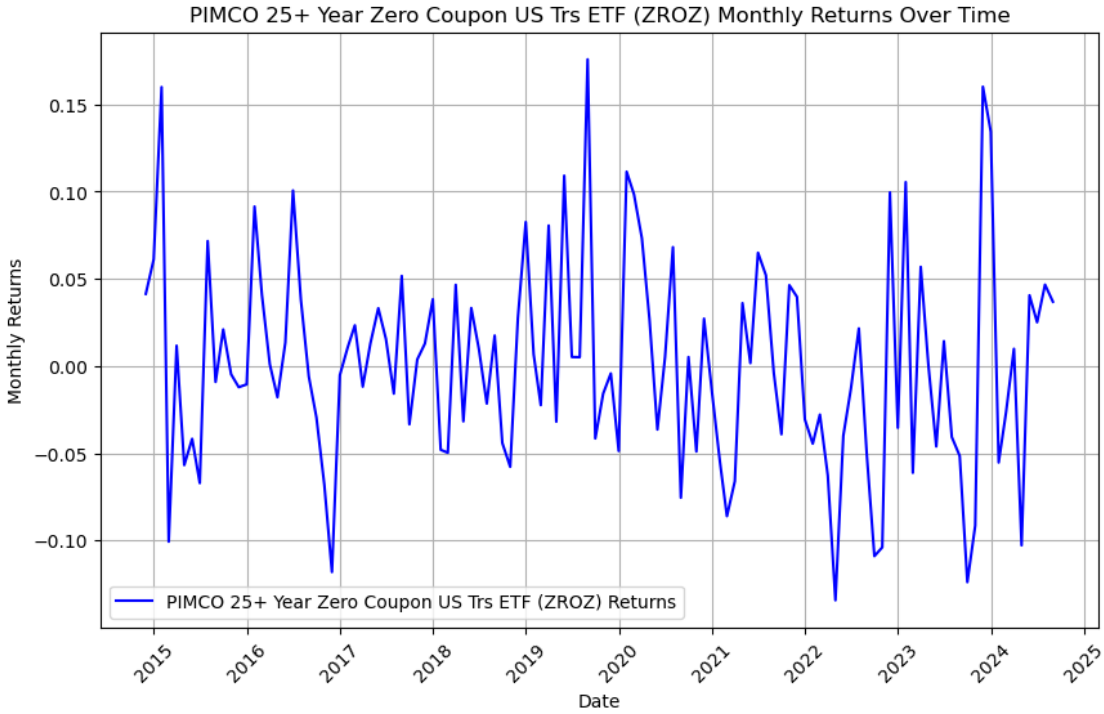
7 Monthly Return Time Series (Figure 5)

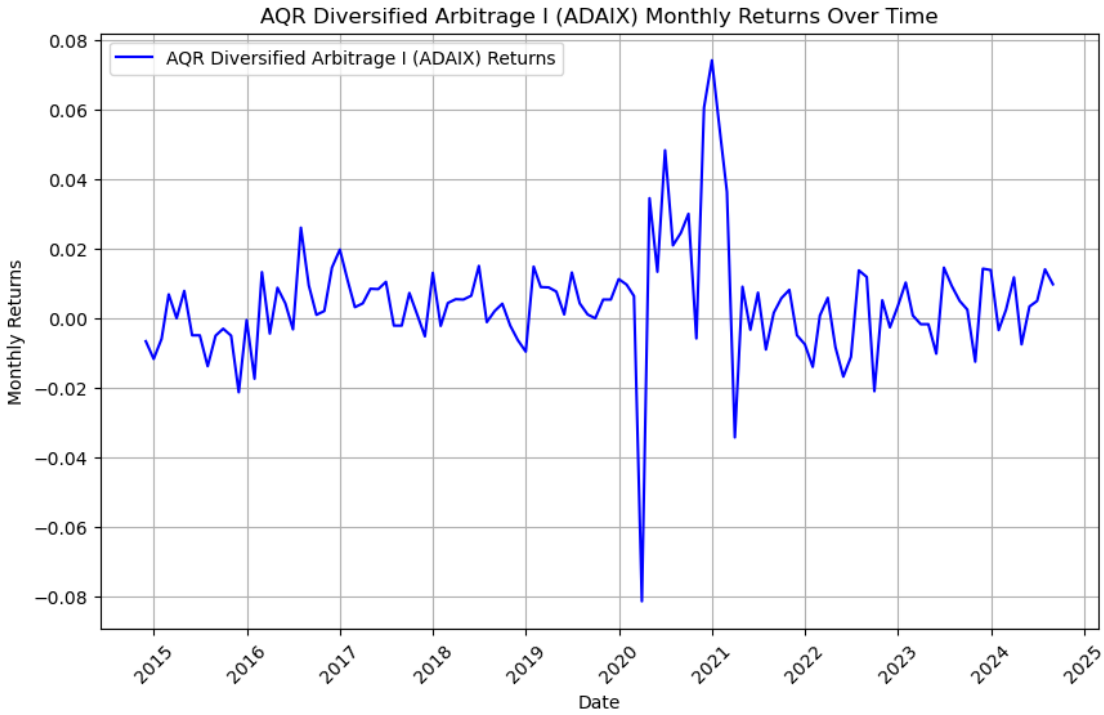
[11]:
```
import os
import matplotlib.pyplot as plt
from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
from matplotlib.figure import Figure
import numpy as np
import math  # Added to calculate rows and columns

# 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)
```

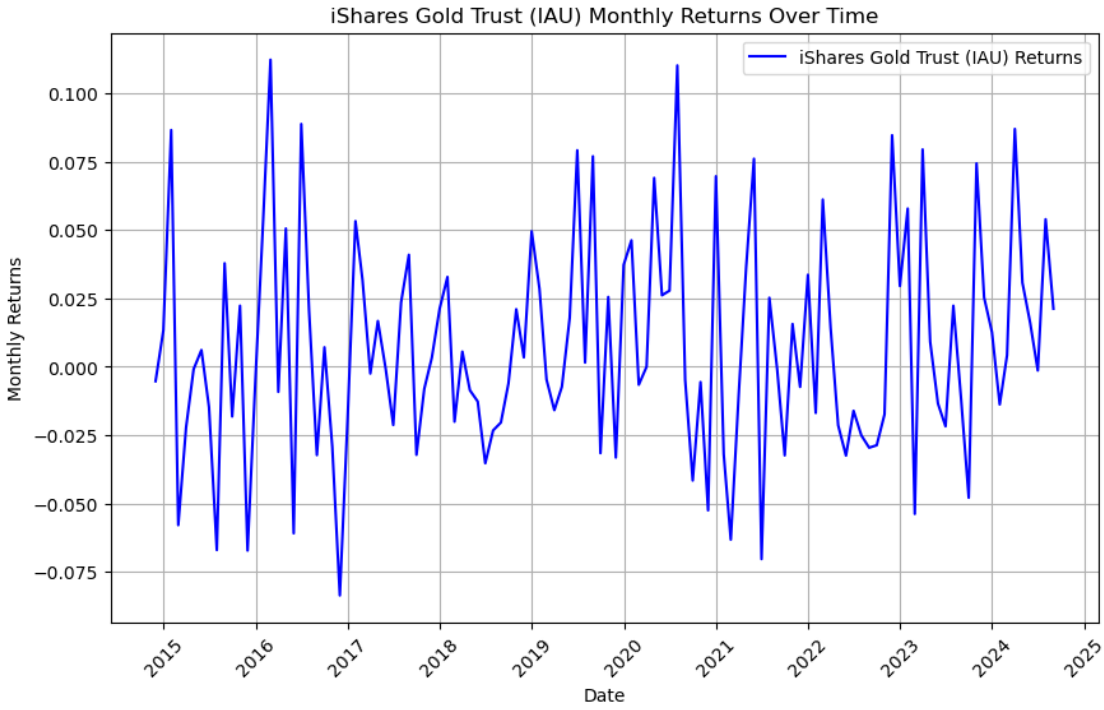
```
num_cols = 3  # You can adjust this to set the number of columns per row
num_rows = math.ceil(num_figures / num_cols)  # Calculate the necessary number␣
 ↪of rows

# 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()
```

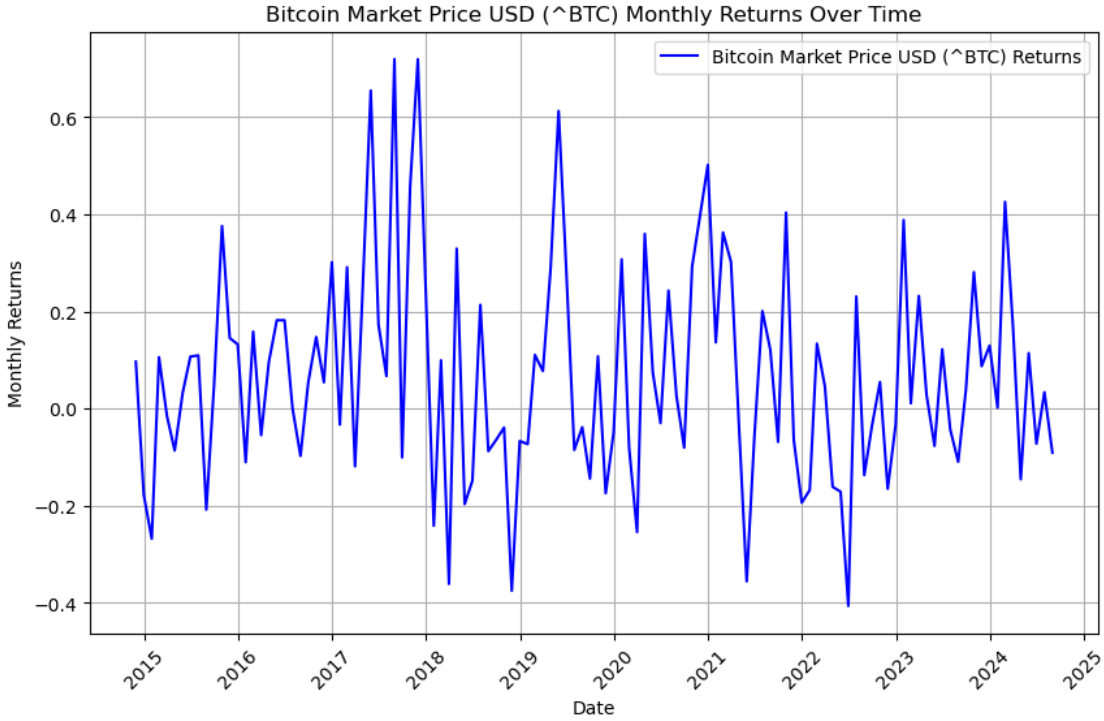

Vanguard LifeStrategy Income Fund (VASIX) Monthly Returns Over Time
Vanguard LifeStrategy Income Fund (VASIX) Returns
Monthly Returns
0.04
0.02
0.00
−0.02
−0.04
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
Date

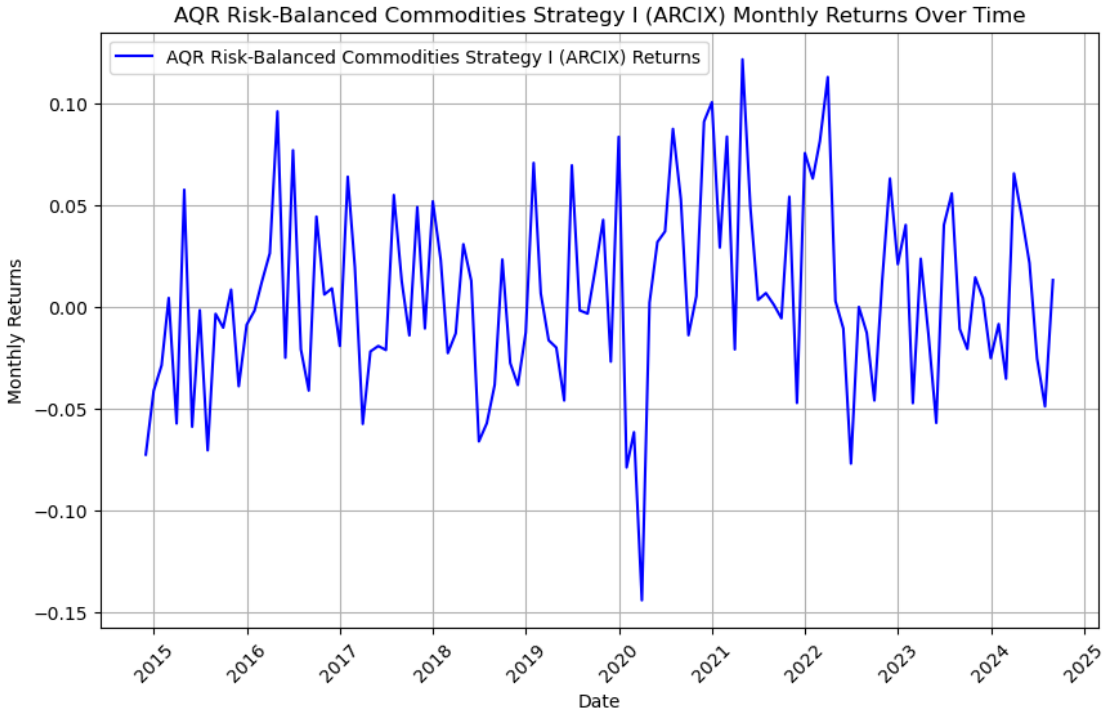

Vanguard Total World Stock ETF (VT) Monthly Returns Over Time
Vanguard Total World Stock ETF (VT) Returns
Monthly Returns
0.10
0.05
0.00
−0.05
−0.10
−0.15
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
Date

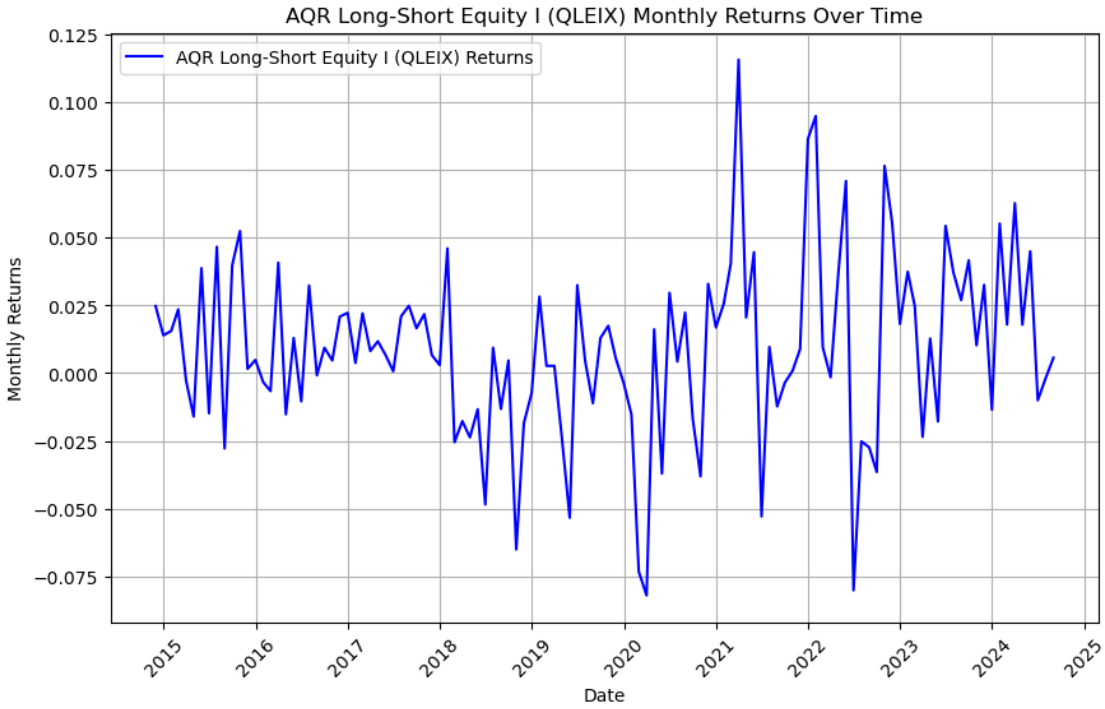

PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ) Monthly Returns Over Time
Monthly Returns
0.15
0.10
0.05
0.00
−0.05
−0.10
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ) Returns
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
Date

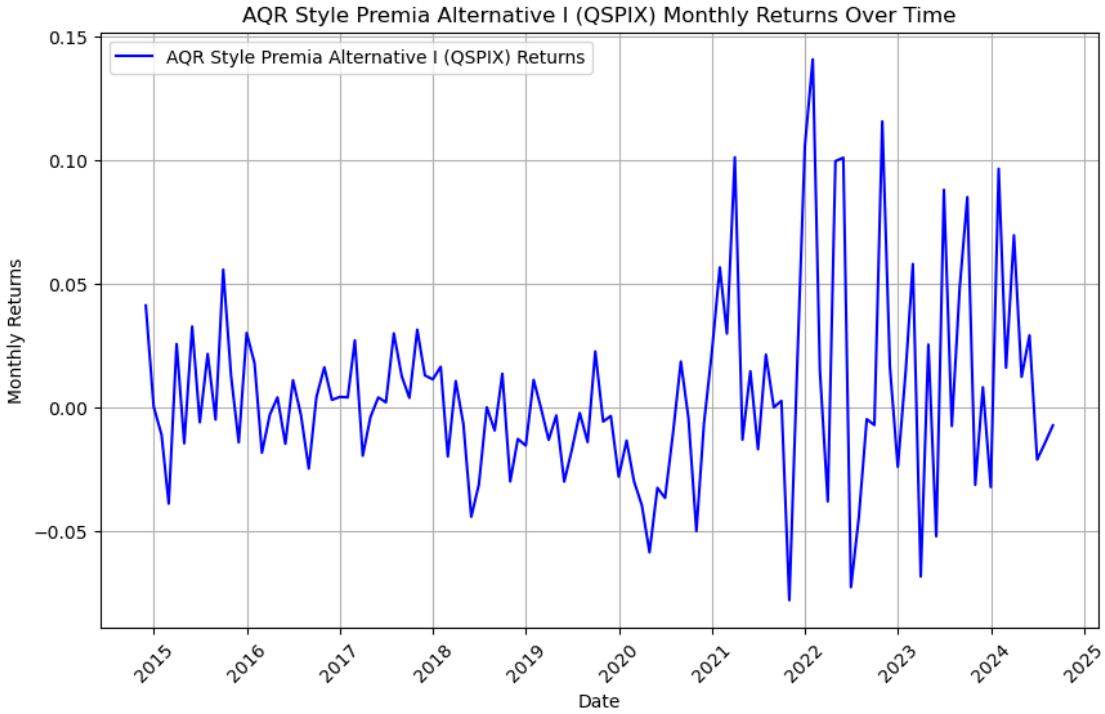

AQR Diversified Arbitrage I (ADAIX) Monthly Returns Over Time
Monthly Returns
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
−0.08
AQR Diversified Arbitrage I (ADAIX) Returns
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
Date

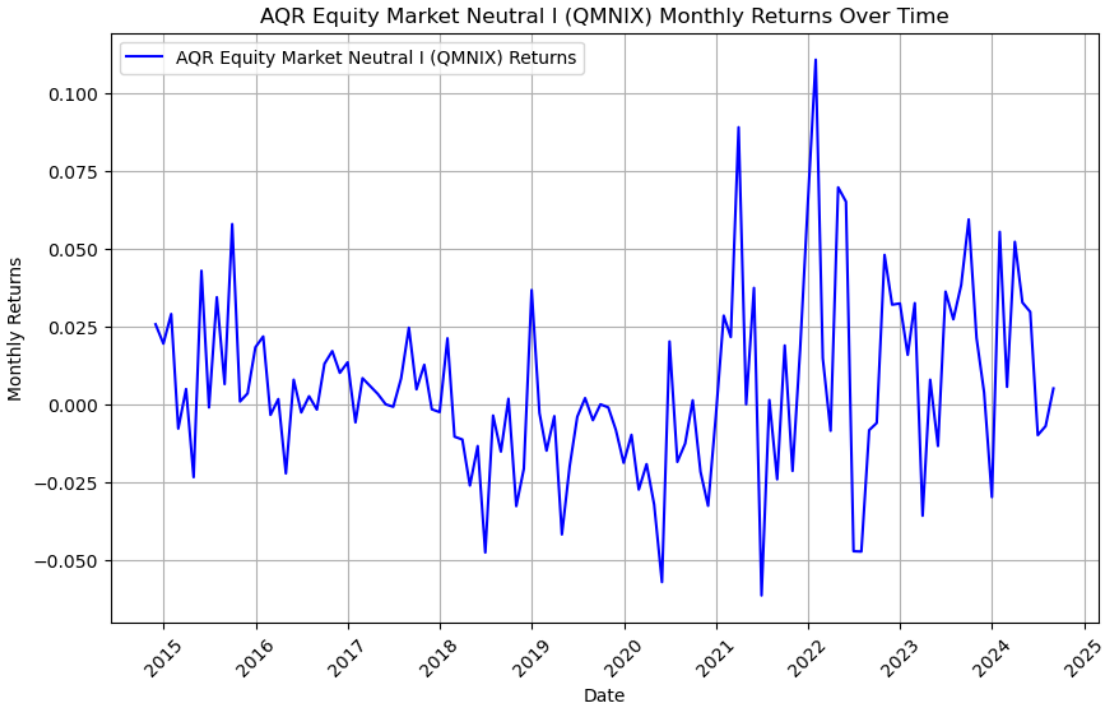

iShares Gold Trust (IAU) Monthly Returns Over Time
iShares Gold Trust (IAU) Returns
Monthly Returns
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
Date

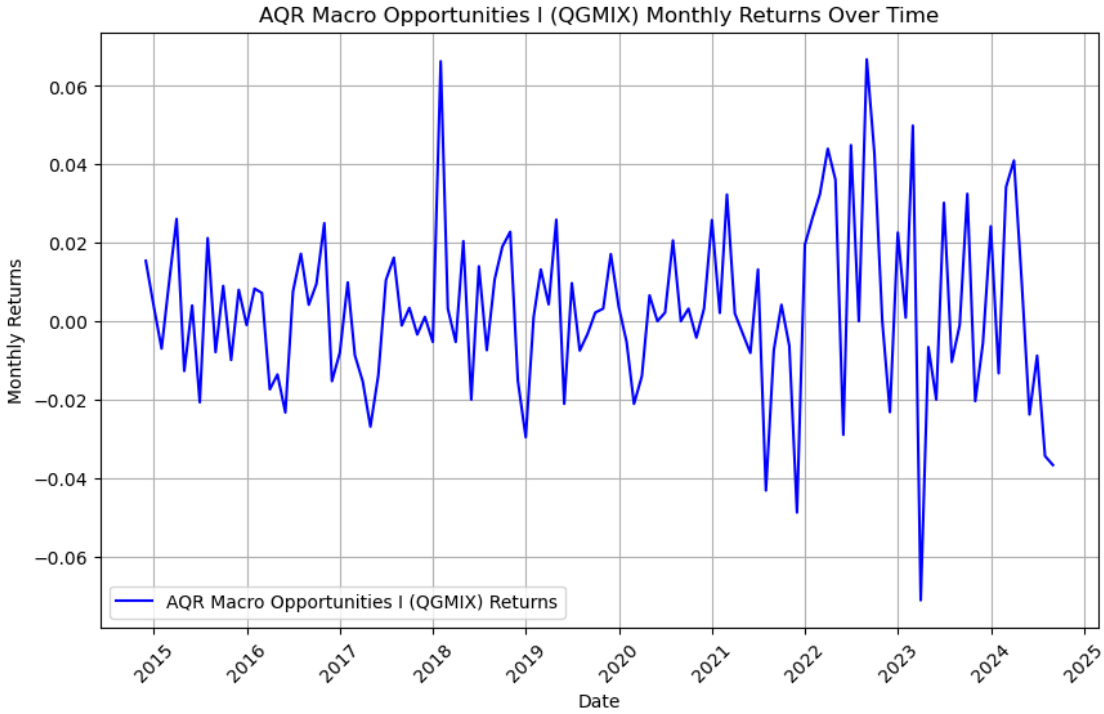

Bitcoin Market Price USD (^BTC) Monthly Returns Over Time
Bitcoin Market Price USD (^BTC) Returns
Monthly Returns
0.6
0.4
0.2
0.0
−0.2
−0.4
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
Date

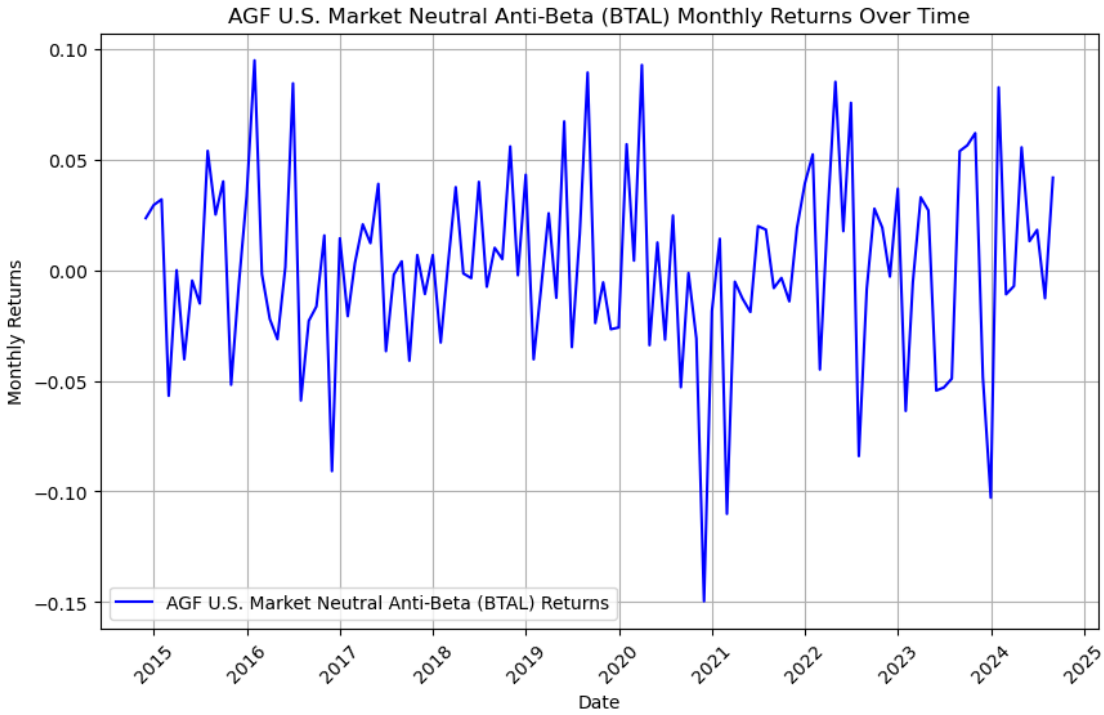

AQR Risk-Balanced Commodities Strategy I (ARCIX) Monthly Returns Over Time
AQR Risk-Balanced Commodities Strategy I (ARCIX) Returns
Monthly Returns
0.10
0.05
0.00
−0.05
−0.10
−0.15
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
Date

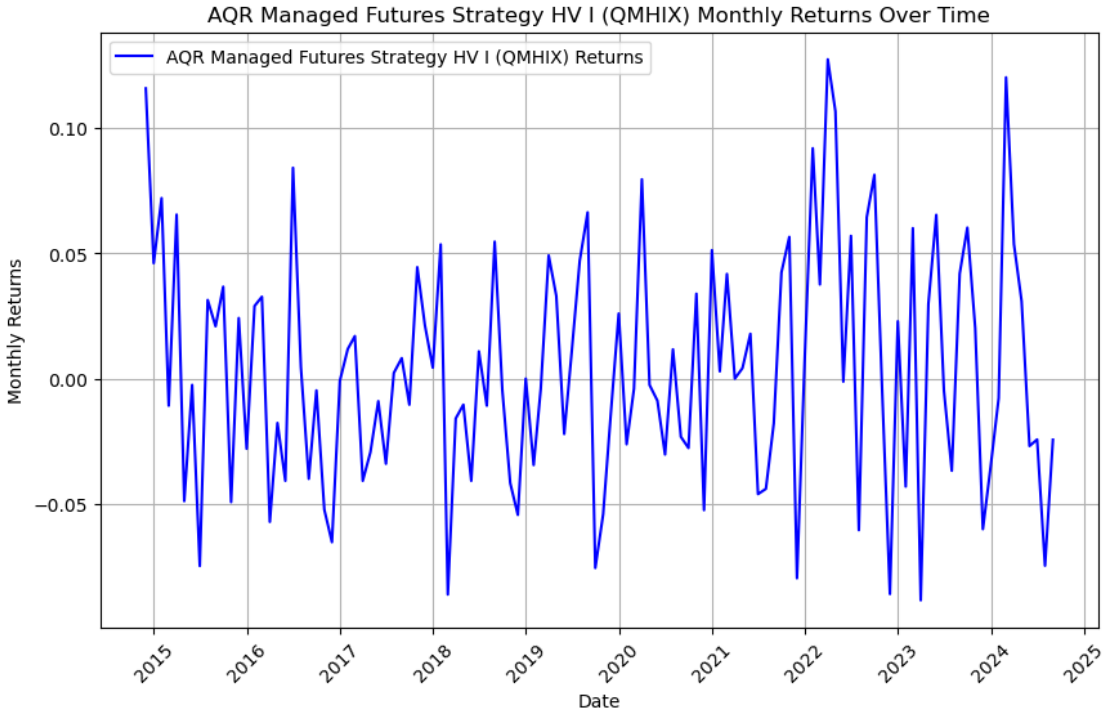

AQR Long-Short Equity I (QLEIX) Monthly Returns Over Time
AQR Long-Short Equity I (QLEIX) Returns
Monthly Returns
0.125
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
Date

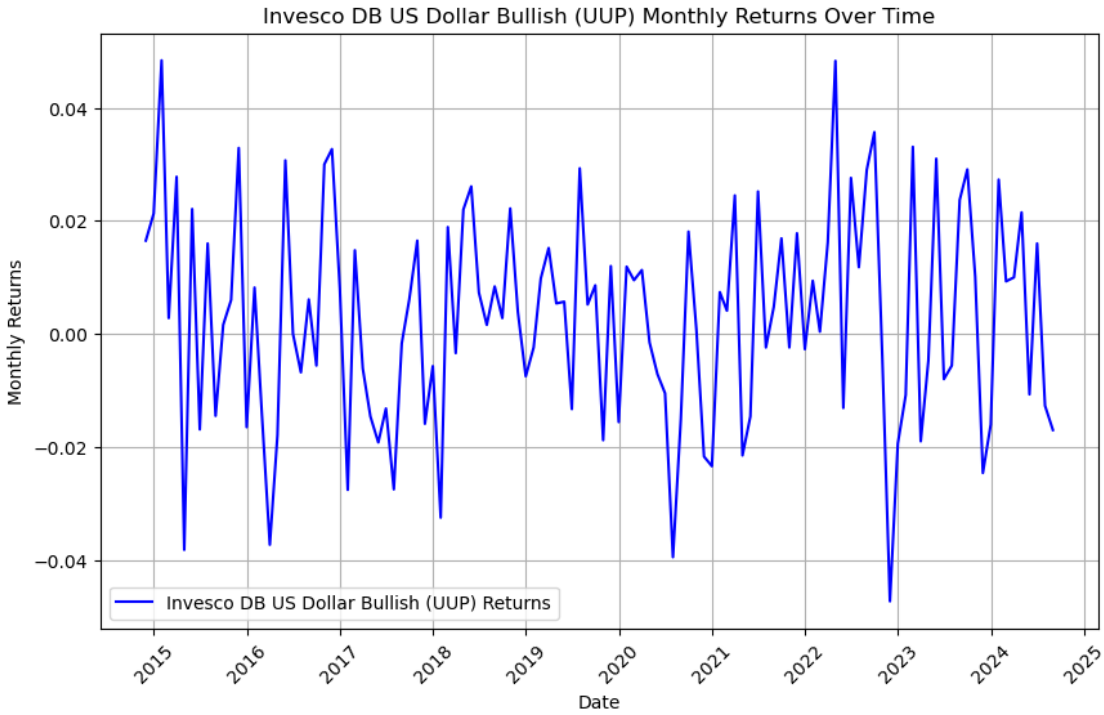

AQR Style Premia Alternative I (QSPIX) Monthly Returns Over Time
AQR Style Premia Alternative I (QSPIX) Returns
Monthly Returns
0.15
0.10
0.05
0.00
−0.05
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
Date

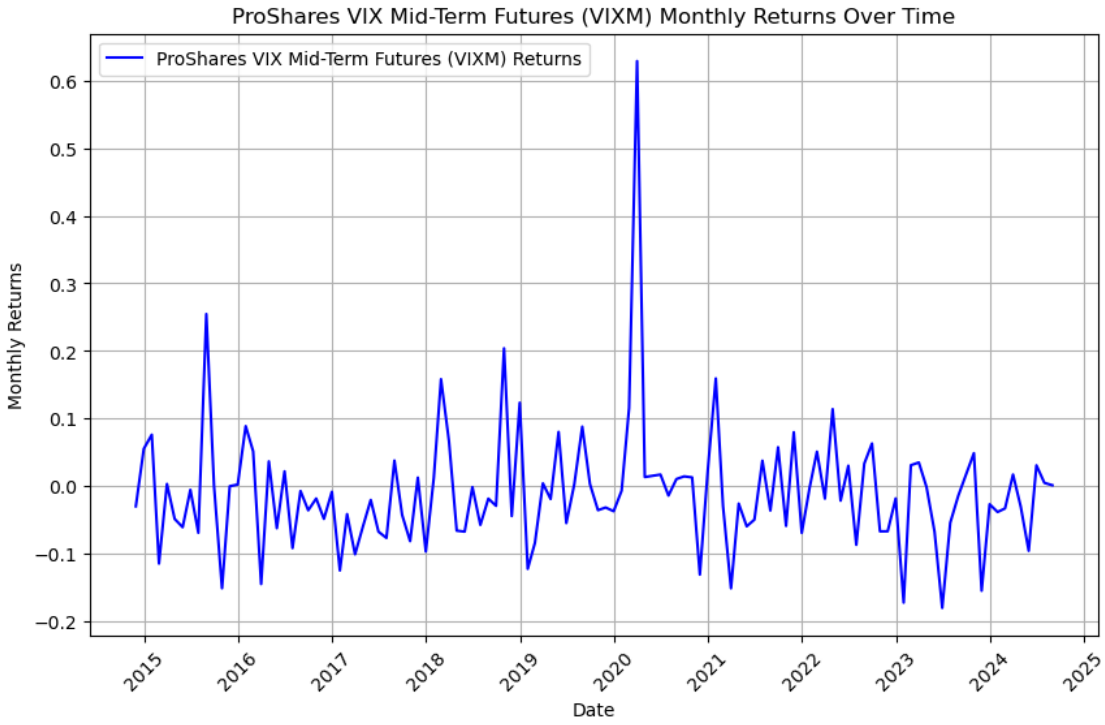

AQR Equity Market Neutral I (QMNIX) Monthly Returns Over Time
AQR Equity Market Neutral I (QMNIX) Returns
Monthly Returns
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
Date

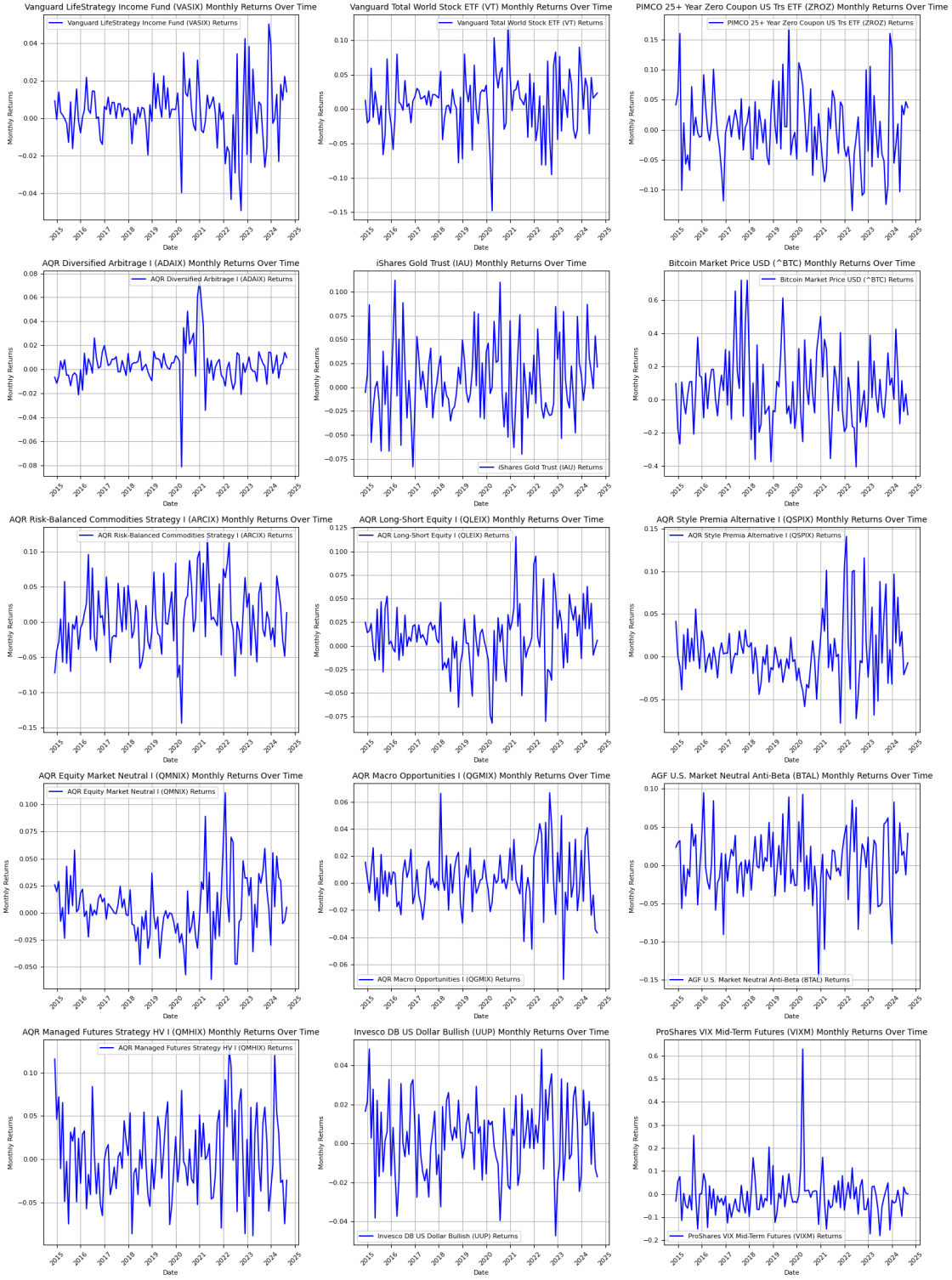

AQR Macro Opportunities I (QGMIX) Monthly Returns Over Time
Monthly Returns
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
AQR Macro Opportunities I (QGMIX) Returns
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
Date


AGF U.S. Market Neutral Anti-Beta (BTAL) Monthly Returns Over Time
Monthly Returns
0.10
0.05
0.00
−0.05
−0.10
−0.15
AGF U.S. Market Neutral Anti-Beta (BTAL) Returns
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
Date


AQR Managed Futures Strategy HV I (QMHIX) Monthly Returns Over Time
AQR Managed Futures Strategy HV I (QMHIX) Returns
Monthly Returns
0.10
0.05
0.00
−0.05
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
Date


Invesco DB US Dollar Bullish (UUP) Monthly Returns Over Time
Invesco DB US Dollar Bullish (UUP) Returns
Monthly Returns
0.04
0.02
0.00
−0.02
−0.04
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
Date

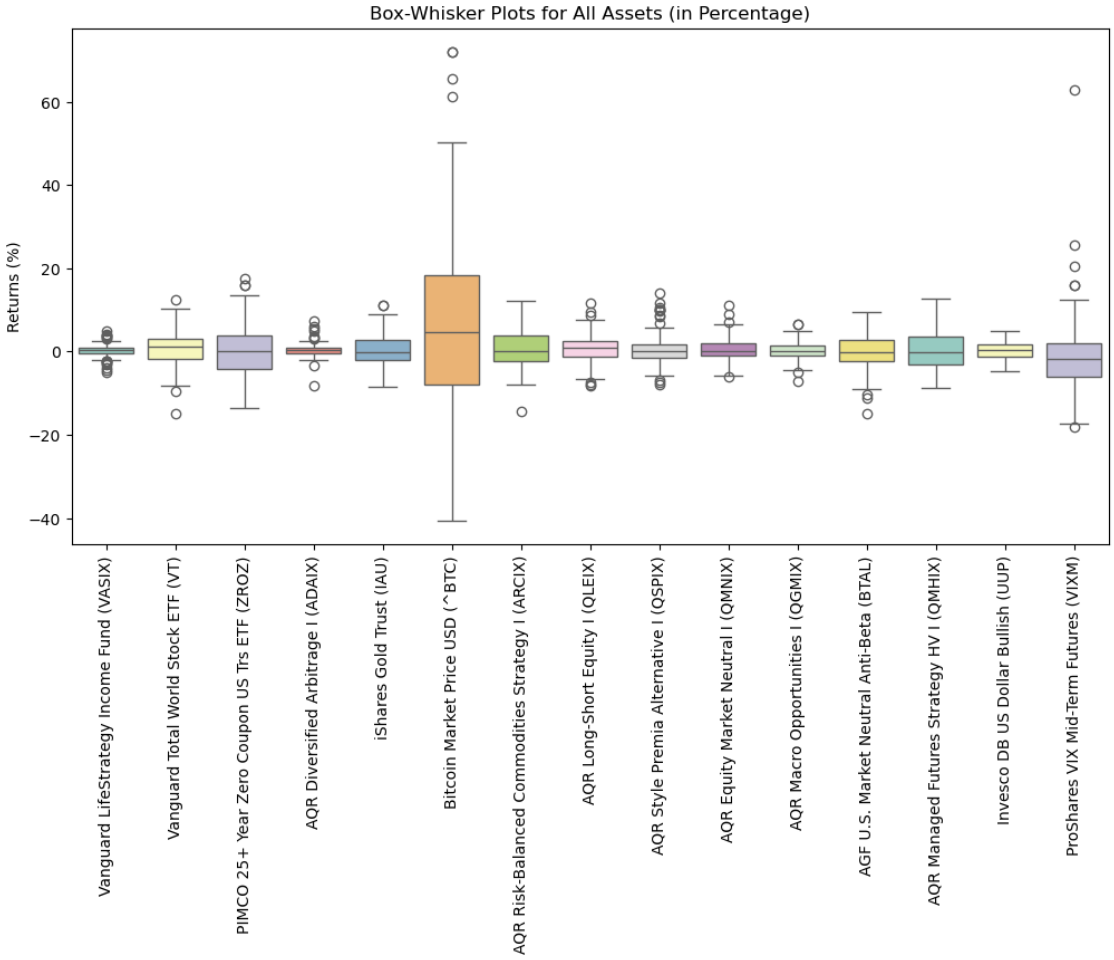

ProShares VIX Mid-Term Futures (VIXM) Monthly Returns Over Time
ProShares VIX Mid-Term Futures (VIXM) Returns
Monthly Returns
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
Date

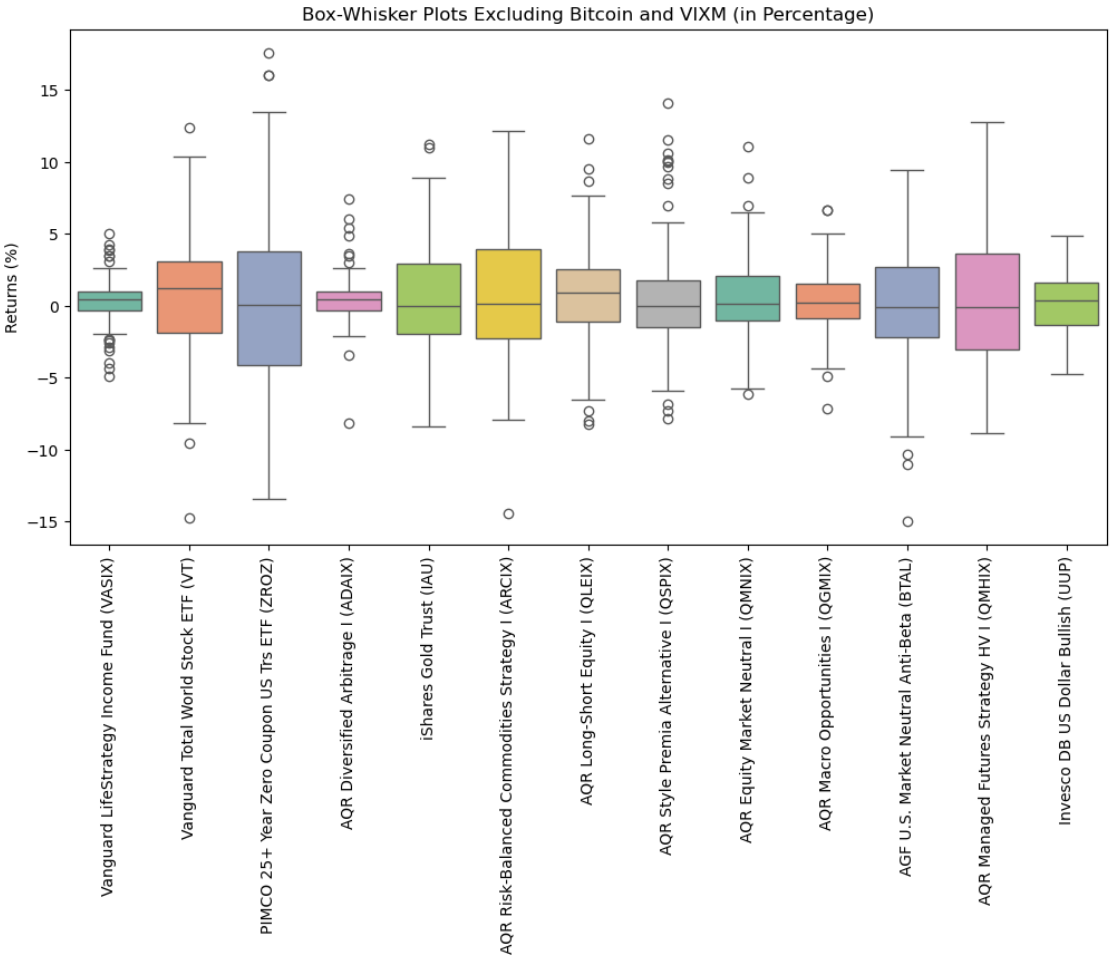

Vanguard LifeStrategy Income Fund (VASIX) Monthly Returns Over Time
Vanguard LifeStrategy Income Fund (VASIX) Returns
Vanguard Total World Stock ETF (VT) Monthly Returns Over Time
Vanguard Total World Stock ETF (VT) Returns
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ) Monthly Returns Over Time
PIMCO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

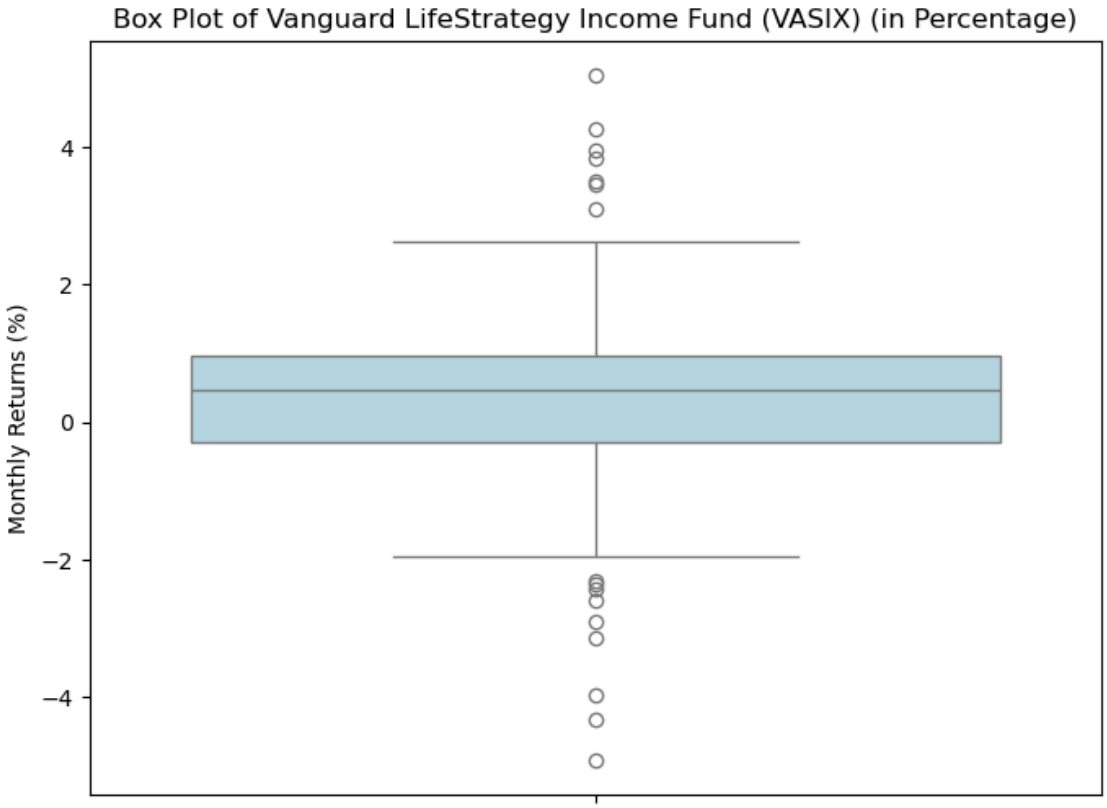
8 Box-Whisker Plots (Figure 6)

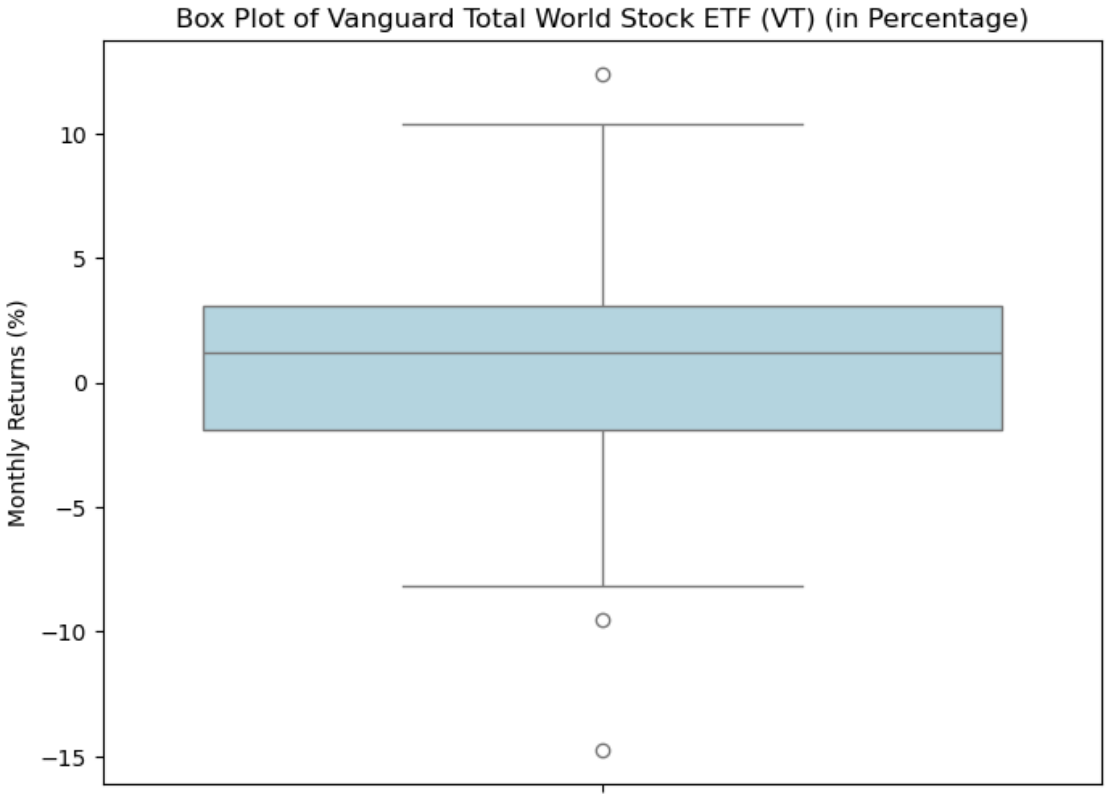
[12]:
```
#BOX AND WHISKER PLOTS

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))
```

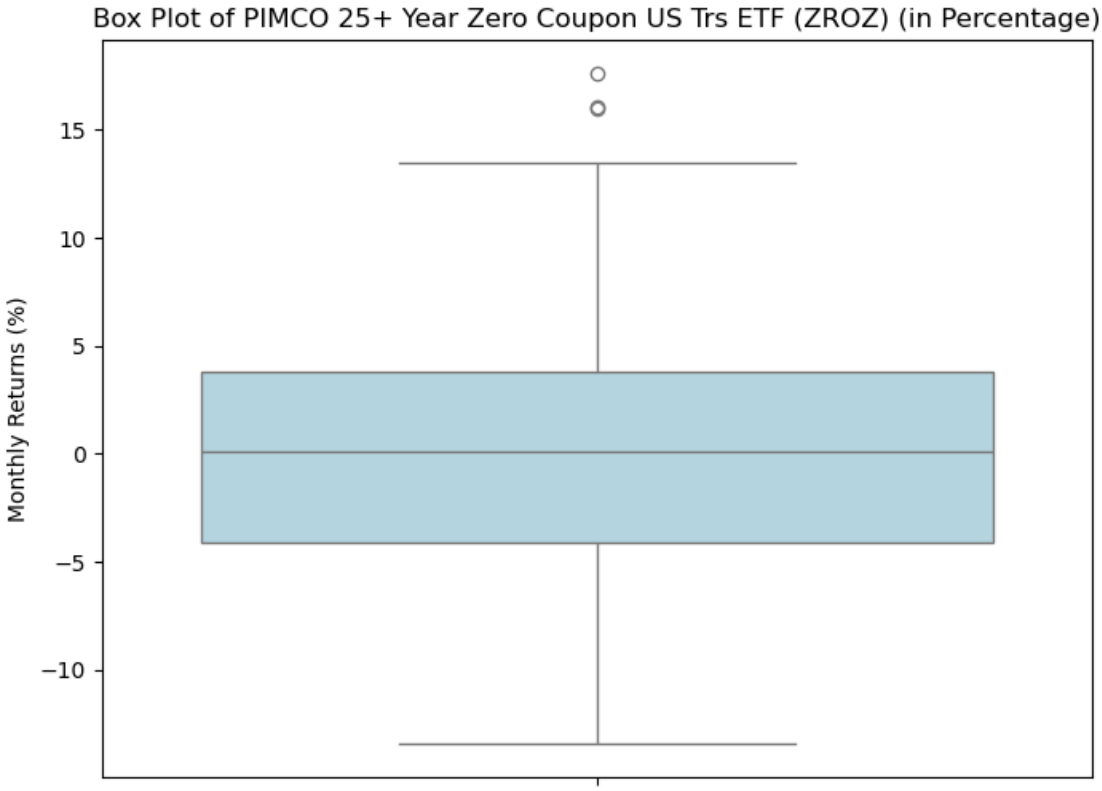
```
    sns.boxplot(data=data_percentage[asset], color='lightblue')
    plt.title(f"Box Plot of {asset} (in Percentage)")
    #plt.xlabel("Monthly Returns (%)")
    plt.ylabel("Monthly Returns (%)")
    plt.savefig(os.path.join(output_folder, f"{asset}_box_whisker.png"),␣
↪bbox_inches='tight', dpi=300)

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

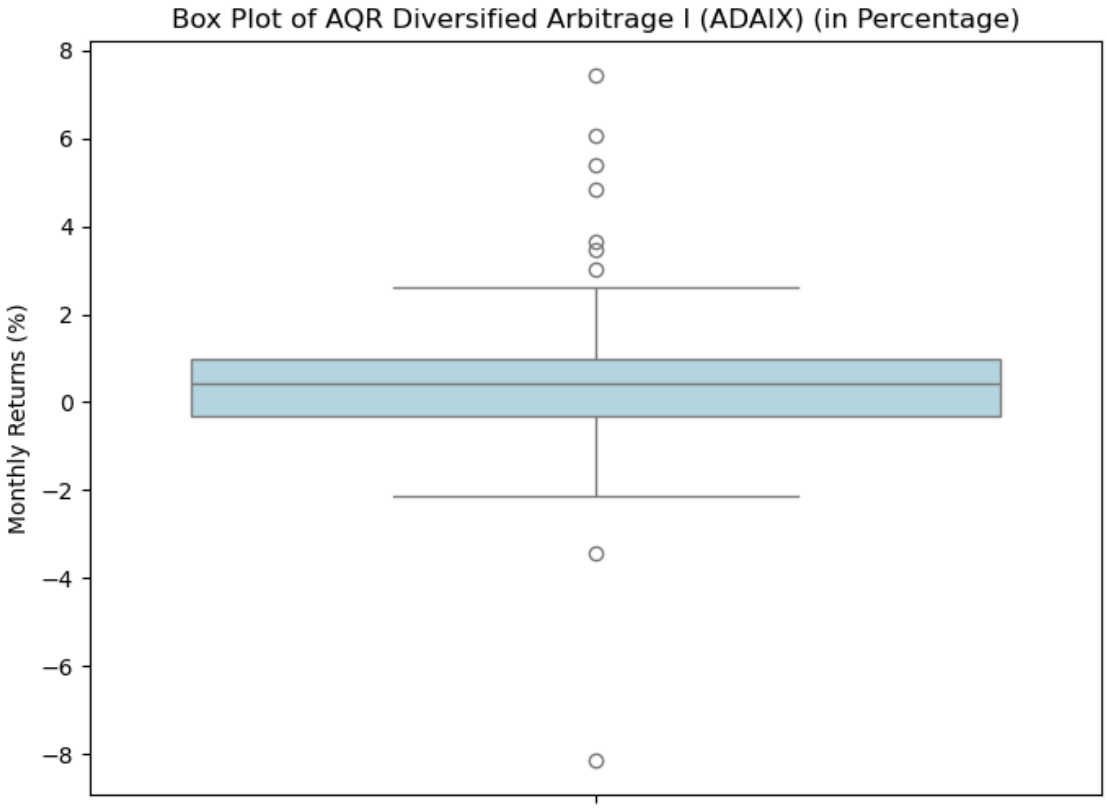


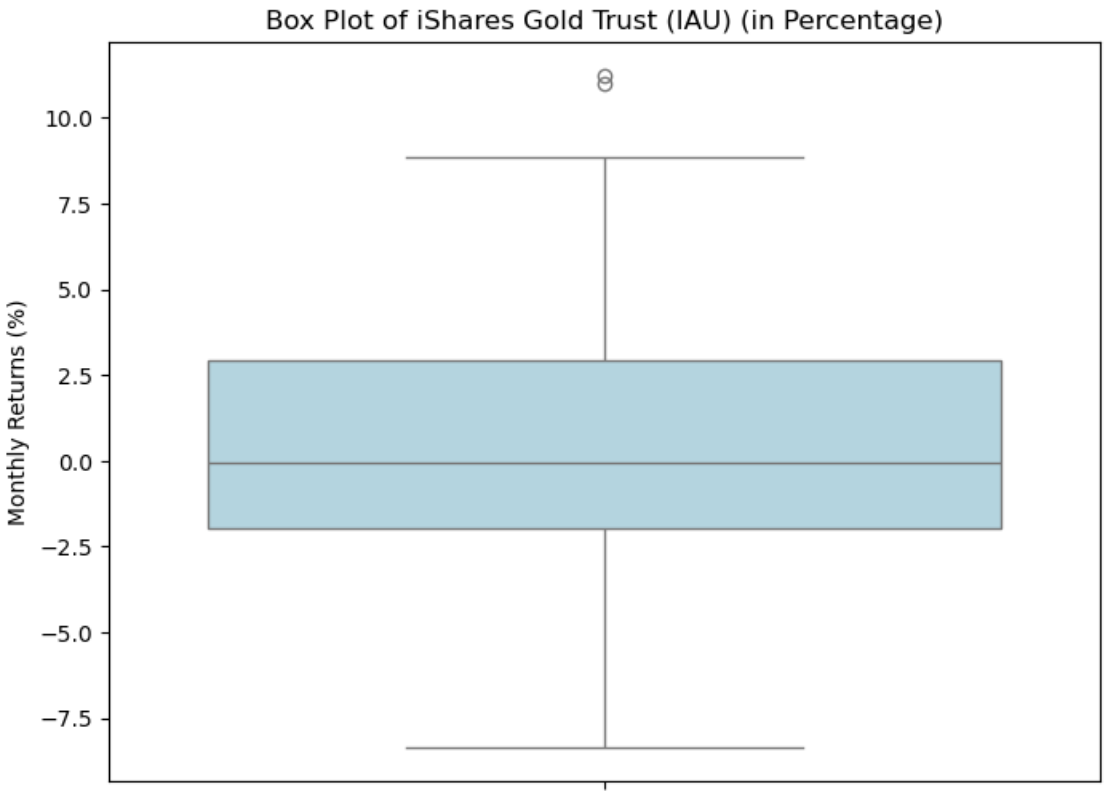

Box-Whisker Plots Excluding Bitcoin and VIXM (in Percentage)
Returns (%)
15
10
5
0
−5
−10
−15
Vanguard LifeStrategy Income Fund (VASIX)
Vanguard Total World Stock ETF (VT)
PIMCO 25+ Year Zero Coupon US Tfs ETF (ZROZ)
AQR Diversified Arbitrage I (ADAIX)
iShares Gold Trust (IAU)
AQR Risk-Balanced Commodities Strategy I (ARCIX)
AQR Long-Short Equity I (QLEIX)
AQR Style Premia Alternative I (QSPIX)
AQR Equity Market Neutral I (QMNIX)
AQR Macro Opportunities I (QGMIX)
AGF U.S. Market Neutral Anti-Beta (BTAL)
AQR Managed Futures Strategy HV I (QMHIX)
Invesco DB US Dollar Bullish (UUP)

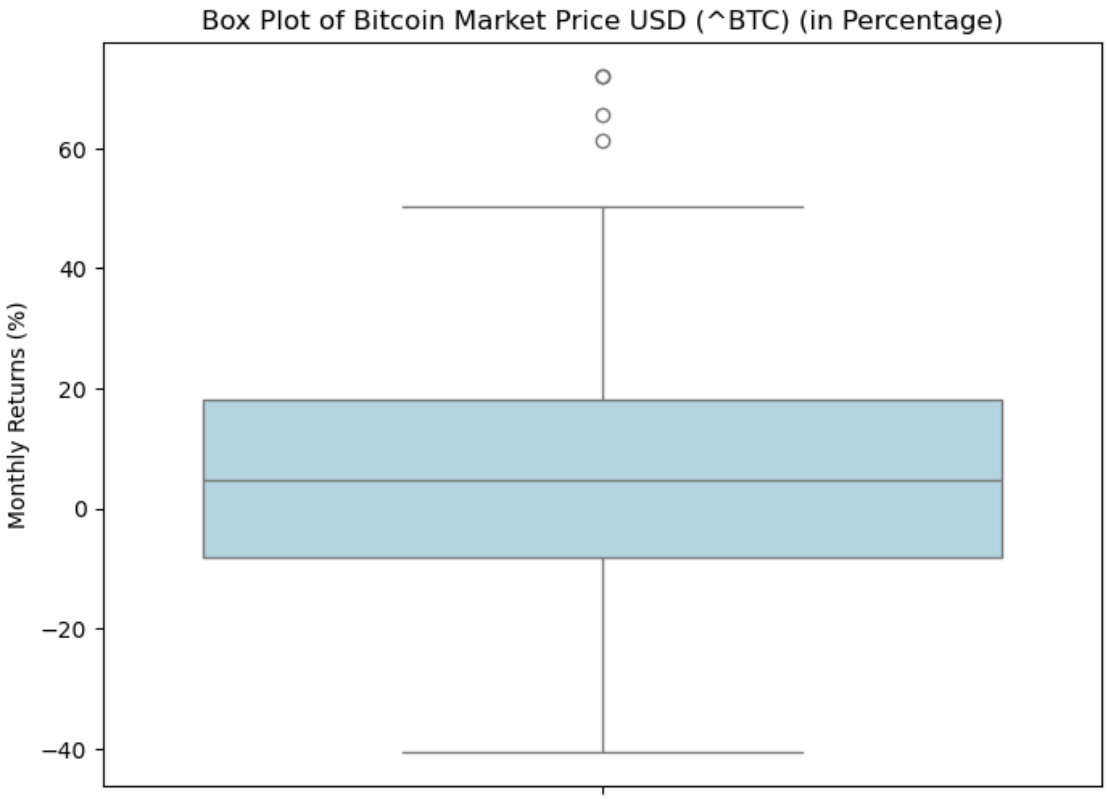

Box Plot of Vanguard LifeStrategy Income Fund (VASIX) (in Percentage)
Monthly Returns (%)
4
2
0
−2
−4

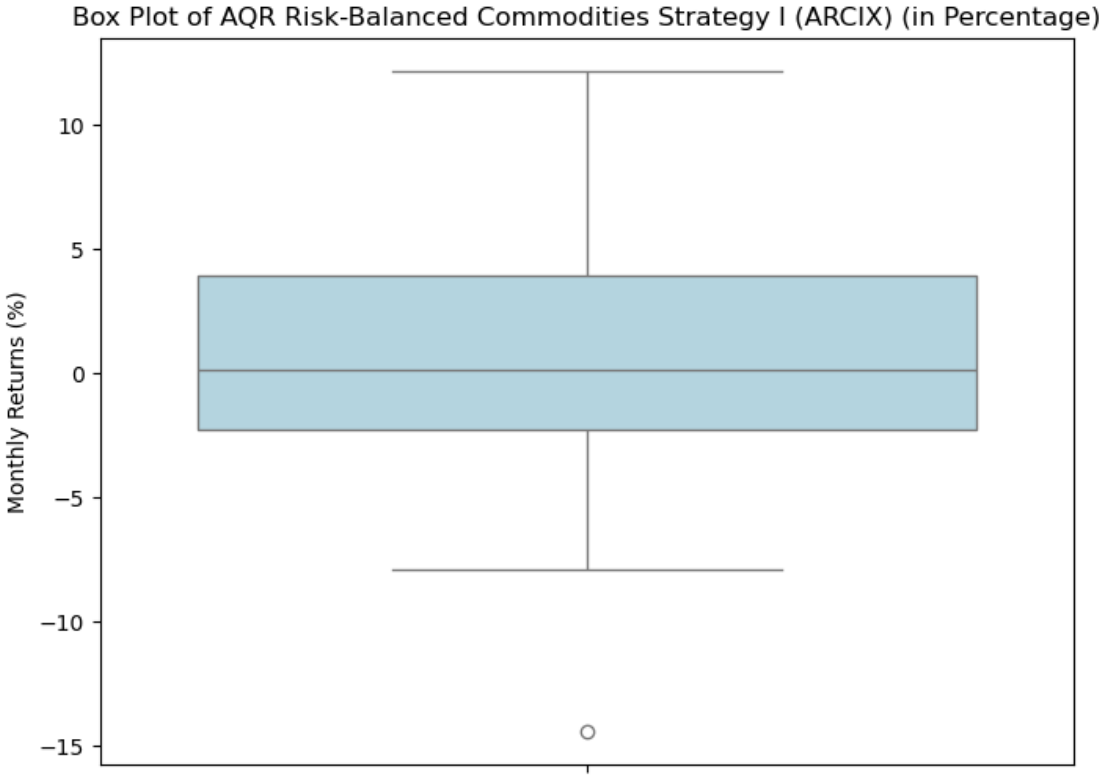

Box Plot of Vanguard Total World Stock ETF (VT) (in Percentage)
Monthly Returns (%)
10
5
0
−5
−10
−15

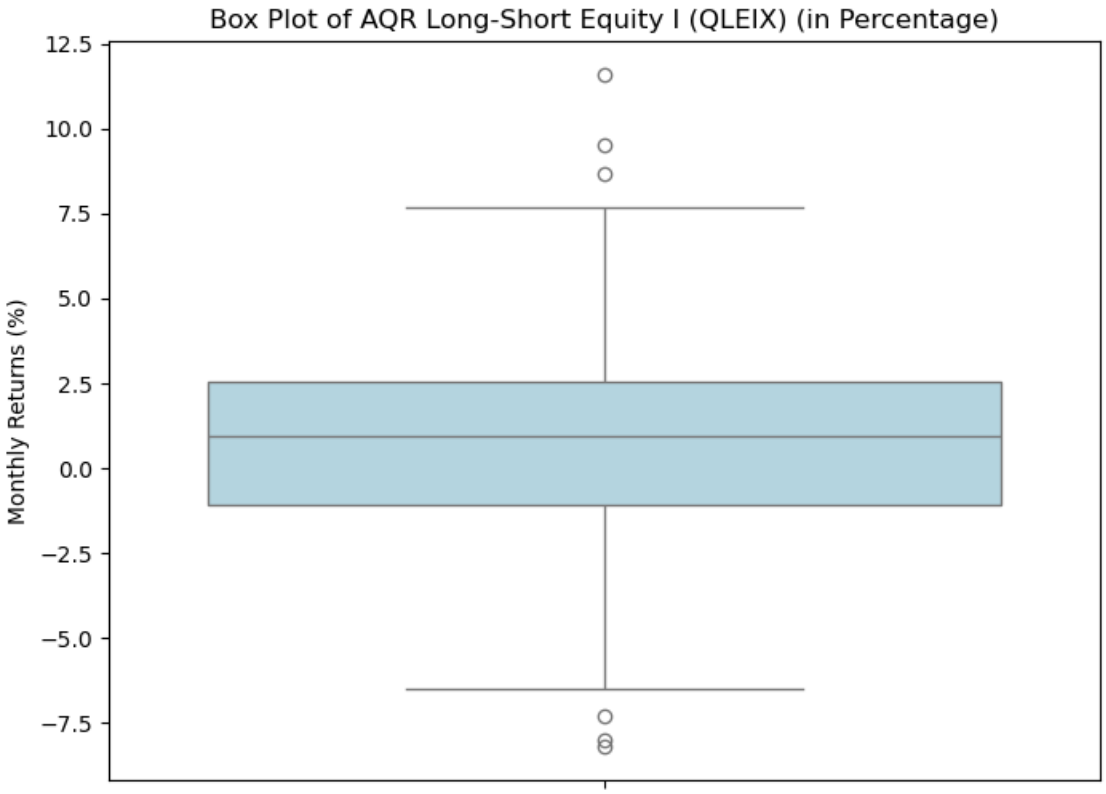

Box Plot of PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ) (in Percentage)
Monthly Returns (%)
15
10
5
0
−5
−10

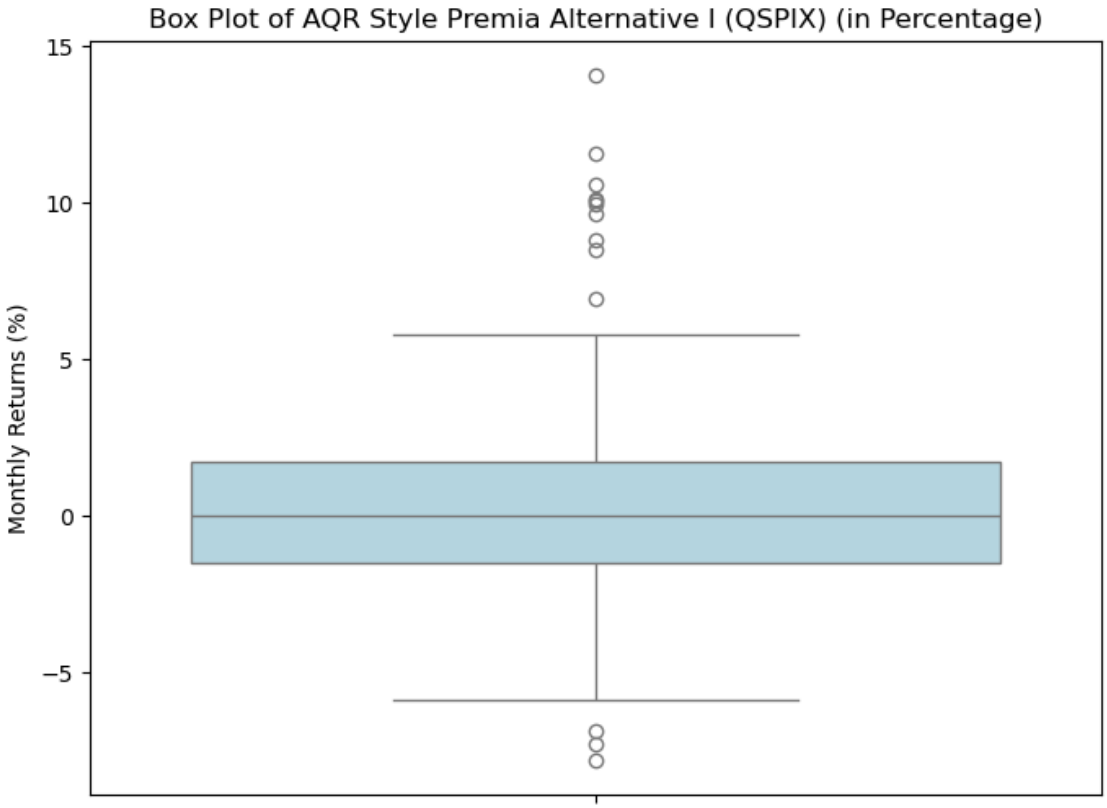

Box Plot of AQR Diversified Arbitrage I (ADAIX) (in Percentage)
Monthly Returns (%)
8
6
4
2
0
−2
−4
−6
−8

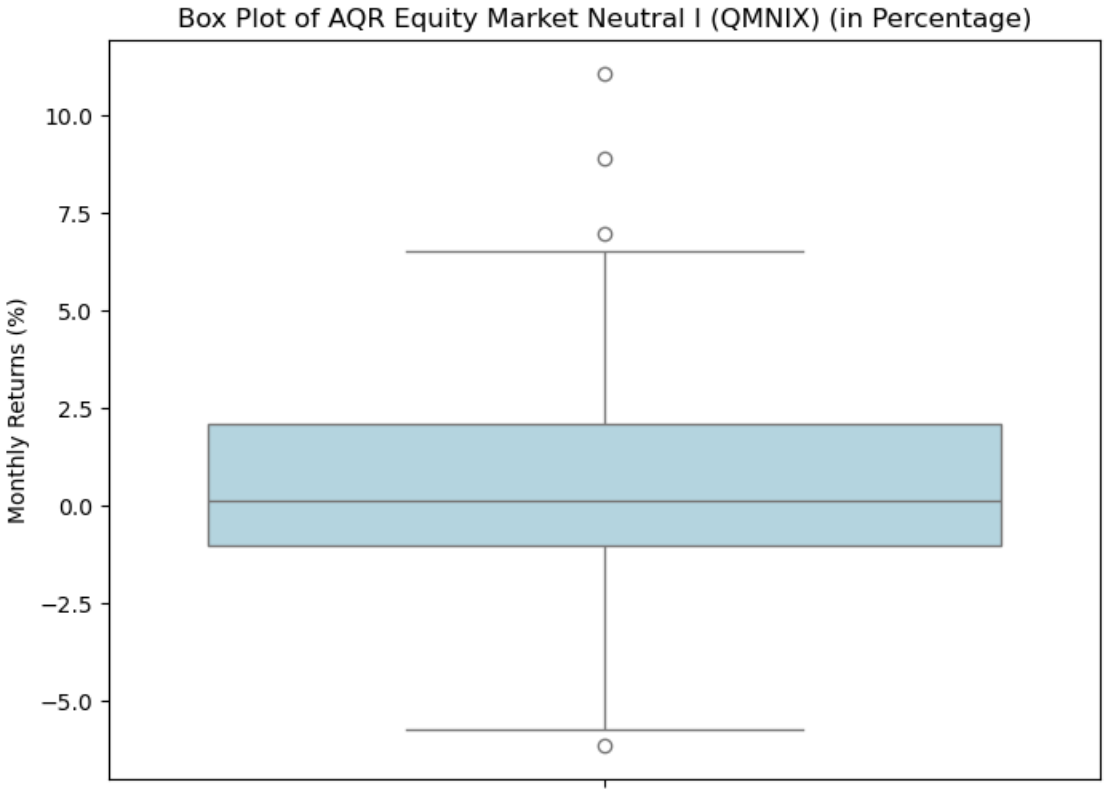

Box Plot of iShares Gold Trust (IAU) (in Percentage)
Monthly Returns (%)
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5

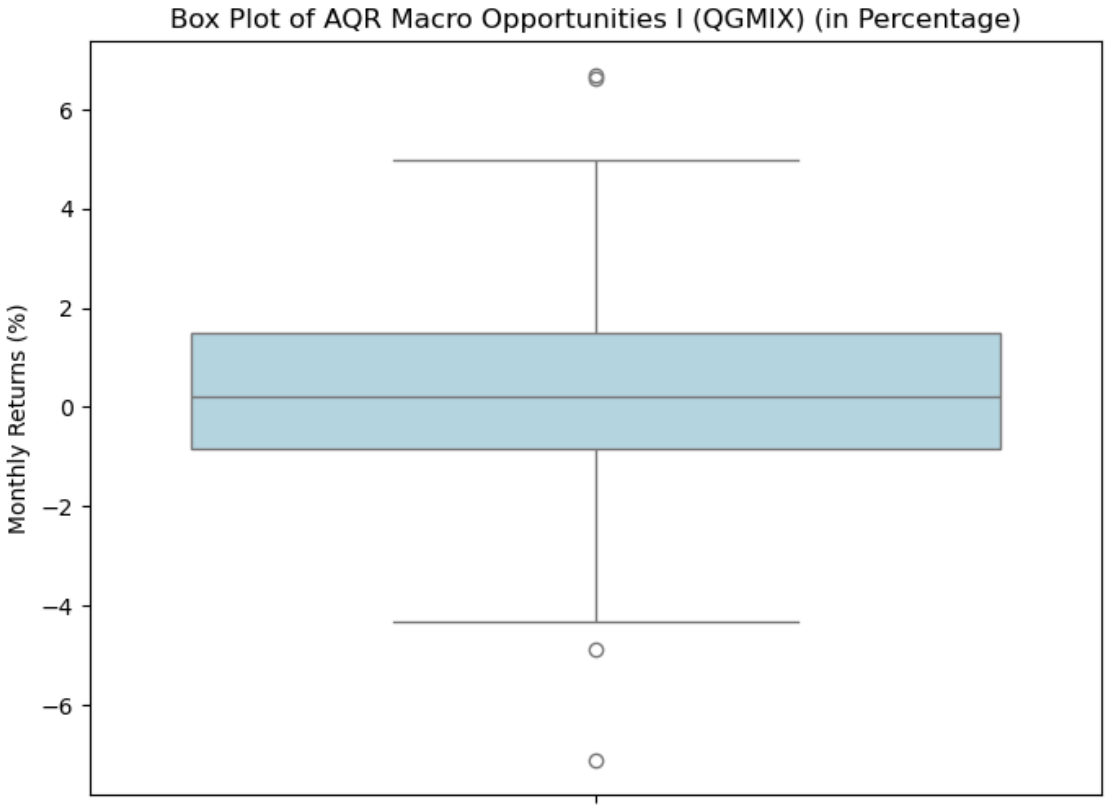

Box Plot of Bitcoin Market Price USD (^BTC) (in Percentage)
Monthly Returns (%)
60
40
20
0
−20
−40

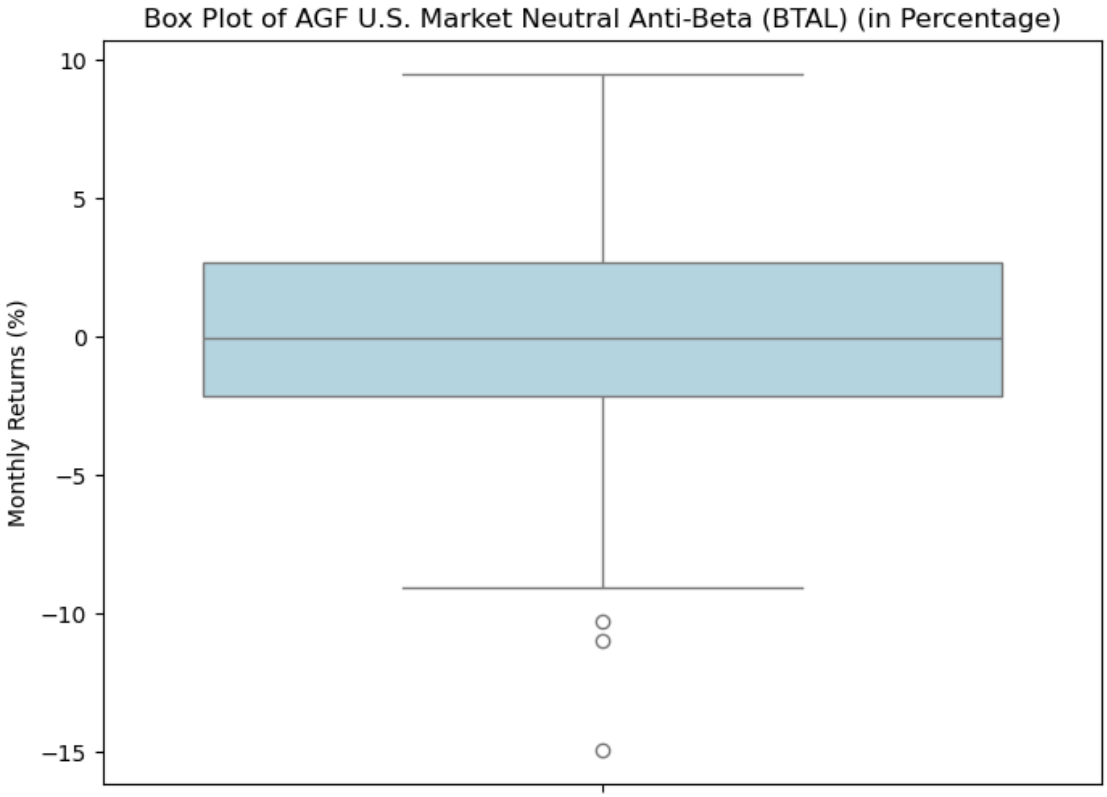

Box Plot of AQR Risk-Balanced Commodities Strategy I (ARCIX) (in Percentage)
Monthly Returns (%)
10
5
0
−5
−10
−15

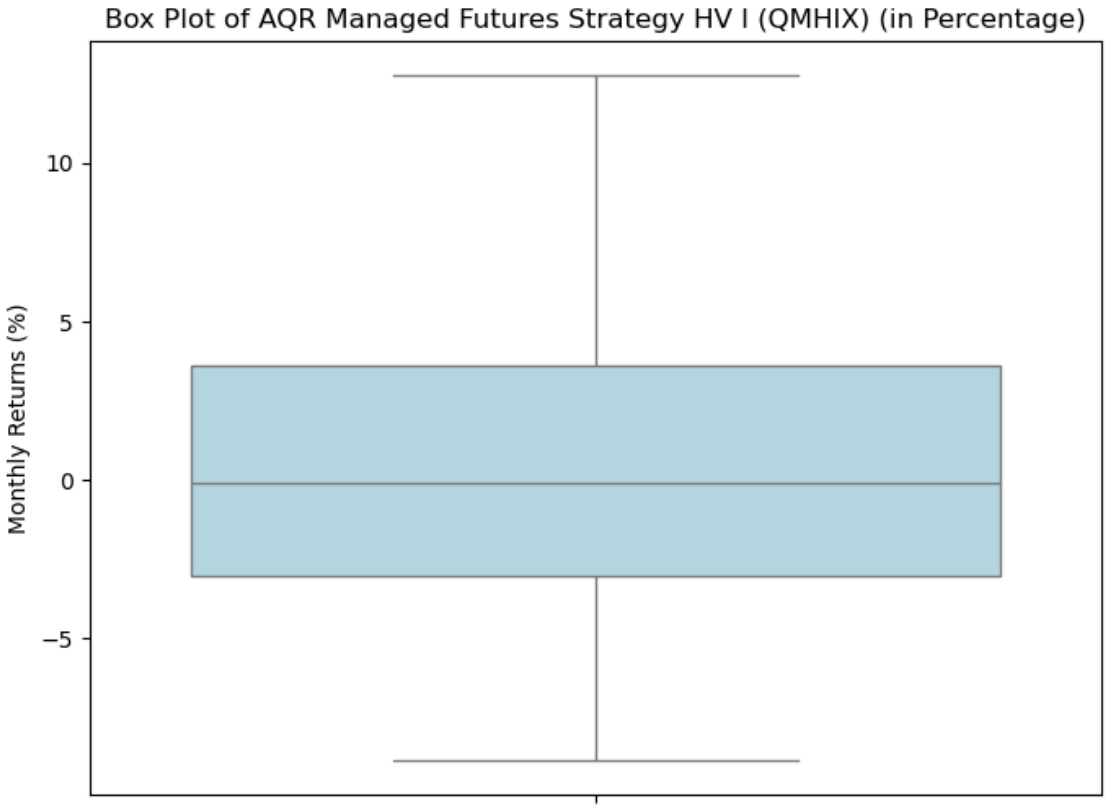

Box Plot of AQR Long-Short Equity I (QLEIX) (in Percentage)
Monthly Returns (%)
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5

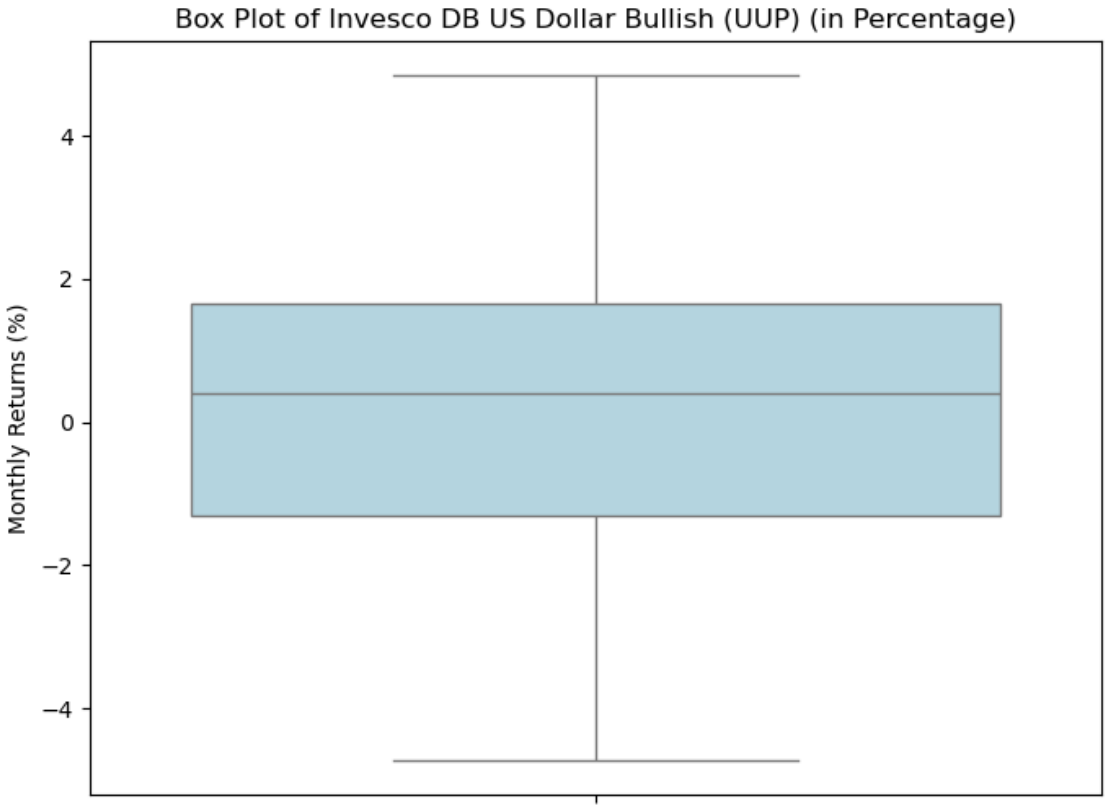

Box Plot of AQR Style Premia Alternative I (QSPIX) (in Percentage)
Monthly Returns (%)
15
10
5
0
−5

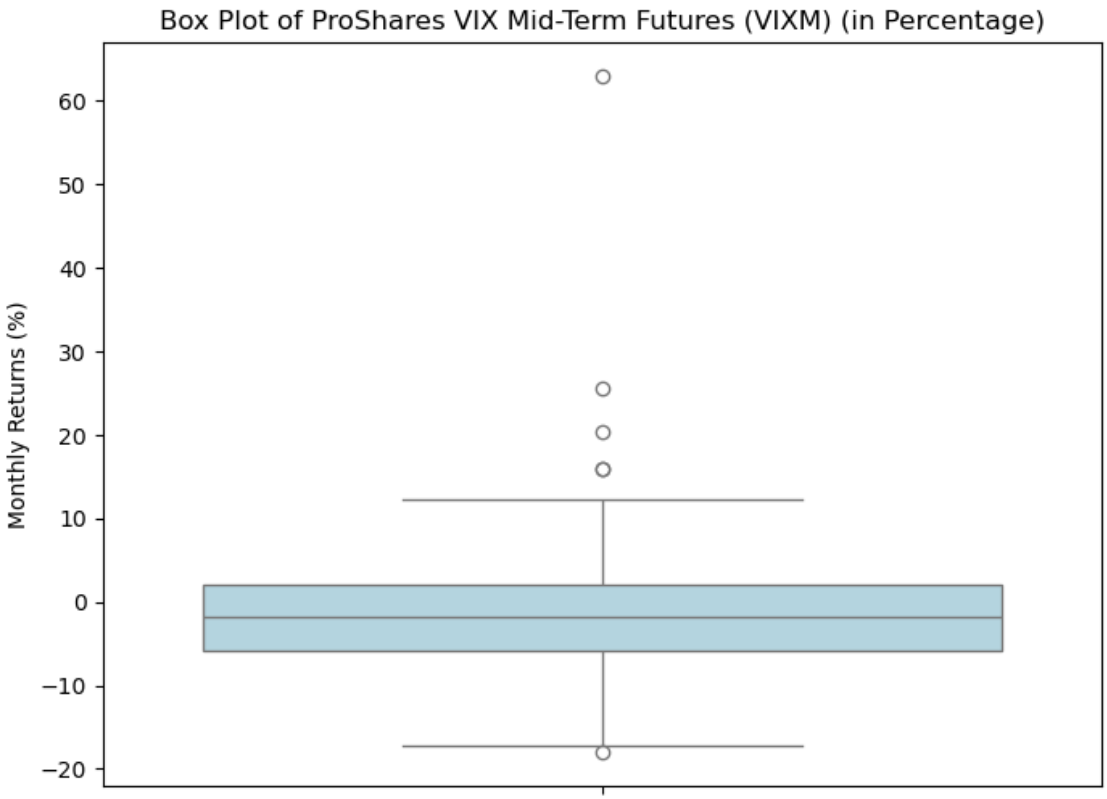

Box Plot of AQR Equity Market Neutral I (QMNIX) (in Percentage)
Monthly Returns (%)
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0


Box Plot of AQR Macro Opportunities I (QGMIX) (in Percentage)
Monthly Returns (%)
6
4
2
0
−2
−4
−6


Box Plot of AGF U.S. Market Neutral Anti-Beta (BTAL) (in Percentage)
Monthly Returns (%)
10
5
0
−5
−10
−15


Box Plot of AQR Managed Futures Strategy HV I (QMHIX) (in Percentage)
Monthly Returns (%)
10
5
0
−5


Box Plot of Invesco DB US Dollar Bullish (UUP) (in Percentage)
Monthly Returns (%)
4
2
0
−2
−4



9 Asset Returns During Benchmark Outlier Months (Table 3)

[13]:
```
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()
```

```
    # 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 (%)"]
```

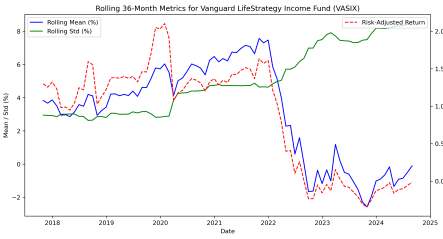
```
# Display the results in a nicely formatted table with percentage formatting␣
 ↪(two decimal places)
bootstrap_results_styled = bootstrap_results_df.style.format({
    "Annualized Mean Estimate": "{:.2f}",
    "Mean 95% CI Lower": "{:.2f}",
    "Mean 95% CI Upper": "{:.2f}",
    "Annualized Std Dev 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
```

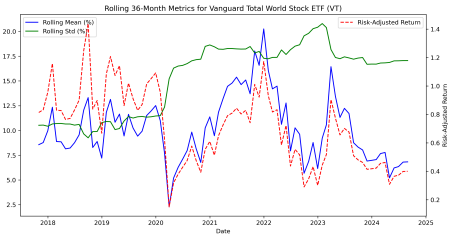
[28]: <pandas.io.formats.style.Styler at 0x169df04d0>

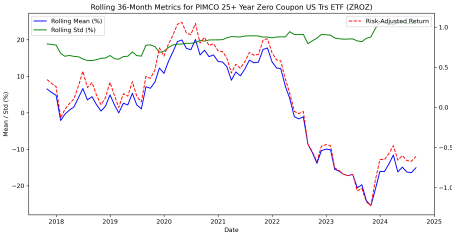
14 Rolling 36-Month Means, Standard Deviations, and Risk-Adj Returns (Figure 7)

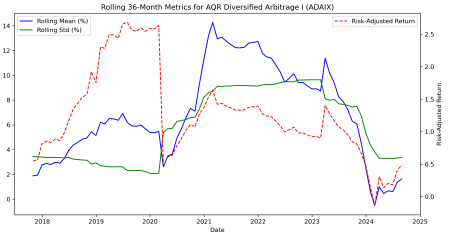
[22]:
```
import os
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from PIL import Image
from IPython.display import display

# 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 (%)')
```

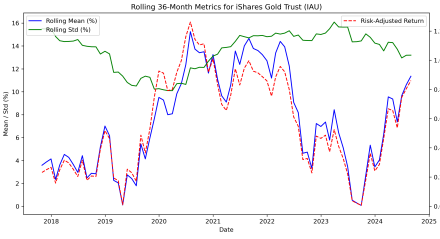
```
    ax1.set_ylabel("Mean / Std (%)")
    ax1.set_xlabel("Date")
    ax1.tick_params(axis='y')
    ax1.legend(loc='upper left')

    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)
```

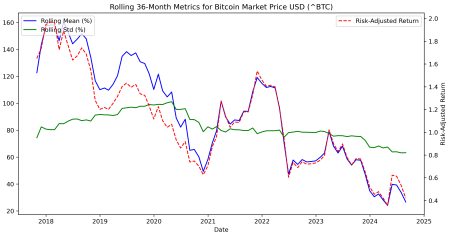

Rolling 36-Month Metrics for Vanguard LifeStrategy Income Fund (VASIX)
Rolling Mean (%)
Rolling Std (%)
Risk-Adjusted Return
Mean / Std (%)
Risk-Adjusted Return
Date

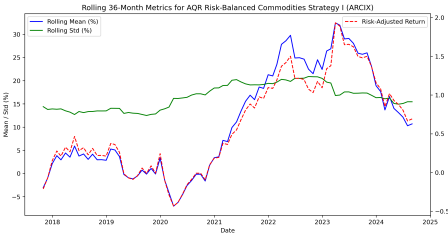

Rolling 36-Month Metrics for Vanguard Total World Stock ETF (VT)
Rolling Mean (%)
Rolling Std (%)
Risk-Adjusted Return
Mean / Std (%)
Risk-Adjusted Return
Date

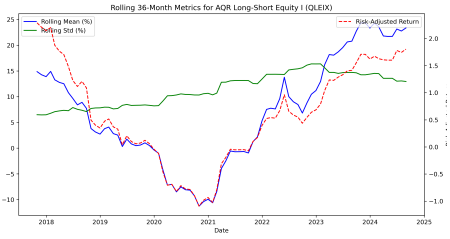

Rolling 36-Month Metrics for PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)
Rolling Mean (%)
Rolling Std (%)
Risk-Adjusted Return
Mean / Std (%)
Risk-Adjusted Return
Date

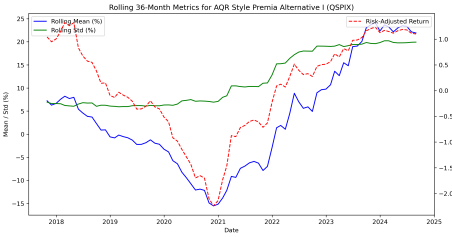

Rolling 36-Month Metrics for AQR Diversified Arbitrage I (ADAIX)
Rolling Mean (%)
Rolling Std (%)
Risk-Adjusted Return
Mean / Std (%)
Risk-Adjusted Return
Date

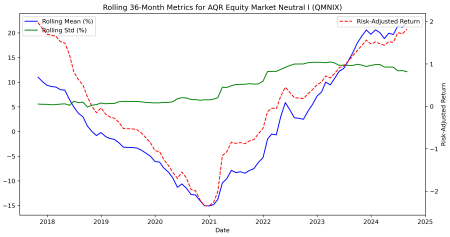

Rolling 36-Month Metrics for iShares Gold Trust (IAU)
Rolling Mean (%)
Rolling Std (%)
Risk-Adjusted Return
Mean / Std (%)
Risk-Adjusted Return
Date

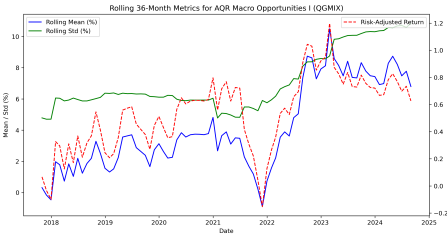

Rolling 36-Month Metrics for Bitcoin Market Price USD (^BTC)
Rolling Mean (%)
Rolling Std (%)
Risk-Adjusted Return
Mean / Std (%)
Risk-Adjusted Return
Date

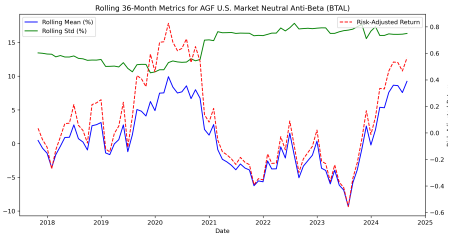

Rolling 36-Month Metrics for AQR Risk-Balanced Commodities Strategy I (ARCIX)
Rolling Mean (%)
Rolling Std (%)
Risk-Adjusted Return
Mean / Std (%)
Risk-Adjusted Return
Date

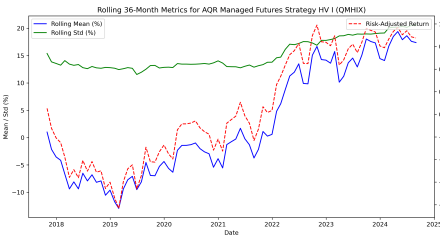

Rolling 36-Month Metrics for AQR Long-Short Equity I (QLEIX)
Rolling Mean (%)
Rolling Std (%)
Risk-Adjusted Return
Mean / Std (%)
Risk-Adjusted Return
Date

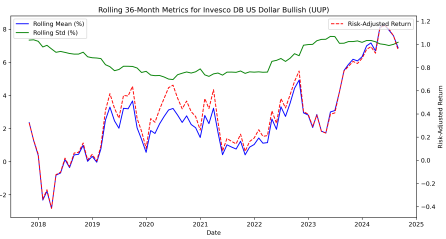

Rolling 36-Month Metrics for AQR Style Premia Alternative I (QSPIX)
Rolling Mean (%)
Rolling Std (%)
Risk-Adjusted Return
Mean / Std (%)
Risk-Adjusted Return
Date

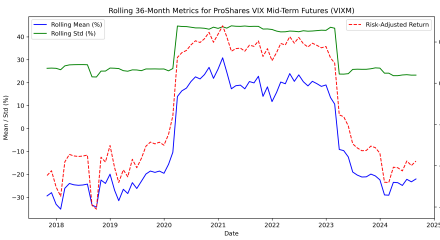

Rolling 36-Month Metrics for AQR Equity Market Neutral I (QMNIX)
Rolling Mean (%)
Rolling Std (%)
Risk-Adjusted Return
Mean / Std (%)
Risk-Adjusted Return
Date


Rolling 36-Month Metrics for AQR Macro Opportunities I (QGMIX)
Rolling Mean (%)
Rolling Std (%)
Risk-Adjusted Return
Mean / Std (%)
Risk-Adjusted Return
Date


Rolling 36-Month Metrics for AGF U.S. Market Neutral Anti-Beta (BTAL)
Rolling Mean (%)
Rolling Std (%)
Risk-Adjusted Return
Mean / Std (%)
Risk-Adjusted Return
Date


Rolling 36-Month Metrics for AQR Managed Futures Strategy HV I (QMHIIX)
Rolling Mean (%)
Rolling Std (%)
Risk-Adjusted Return
Mean / Std (%)
Risk-Adjusted Return
Date


Rolling 36-Month Metrics for Invesco DB US Dollar Bullish (UUP)
Rolling Mean (%)
Rolling Std (%)
Risk-Adjusted Return
Mean / Std (%)
Risk-Adjusted Return
Date

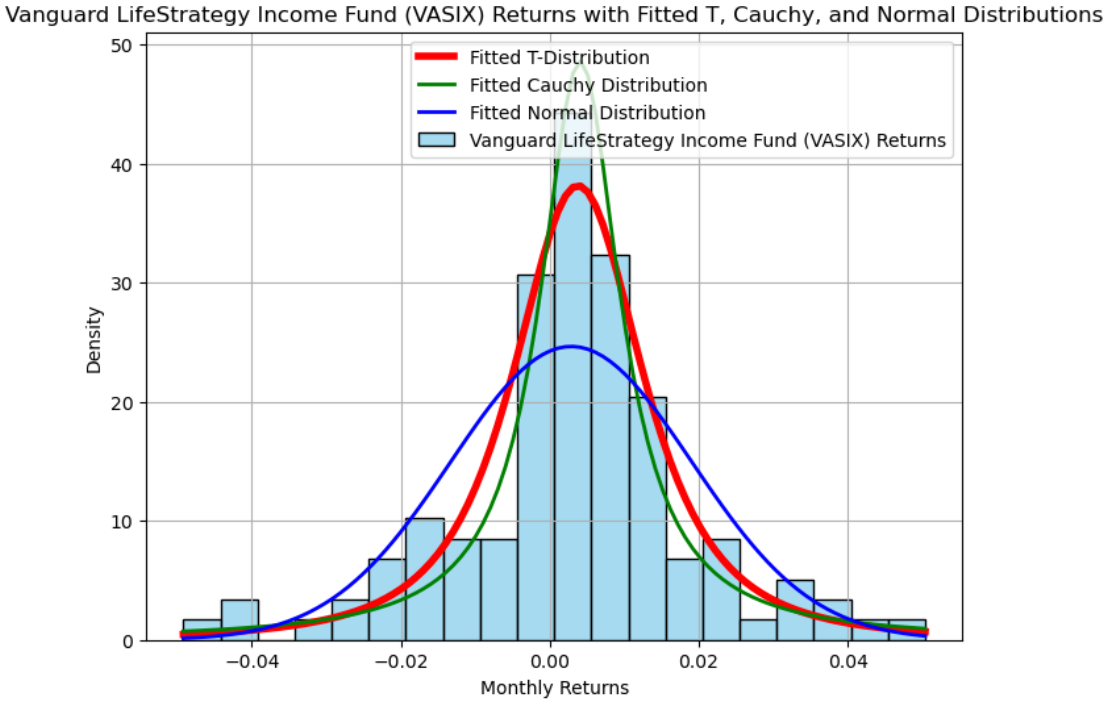

Rolling 36-Month Metrics for ProShares VIX Mid-Term Futures (VIXM)
Rolling Mean (%)
Rolling Std (%)
Risk-Adjusted Return
Mean / Std (%)
Risk-Adjusted Return
Date

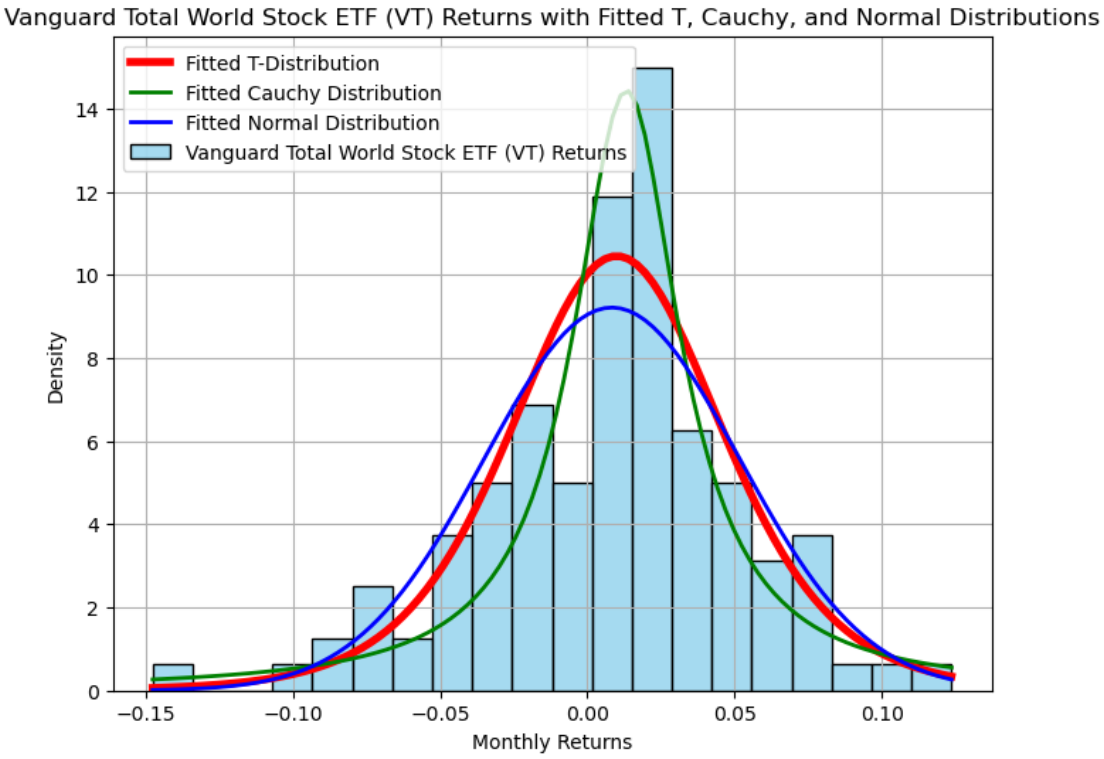
15 Raw Return Histograms with Fitted Probability Distributions (Figure 8)

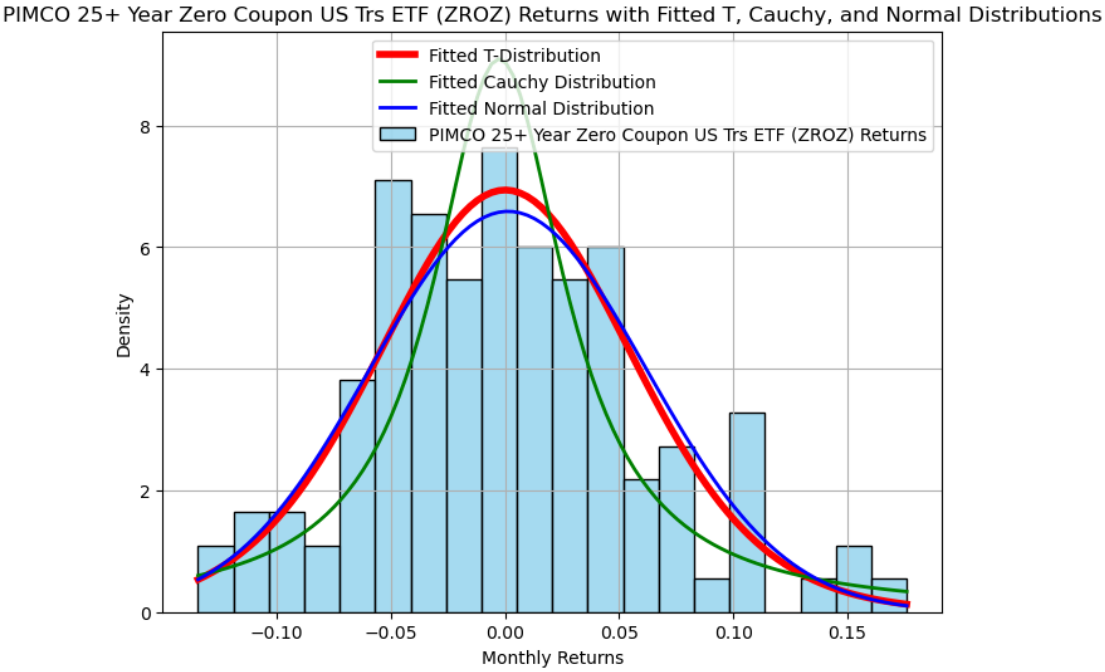
[18]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import t, cauchy, norm

# 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)
```

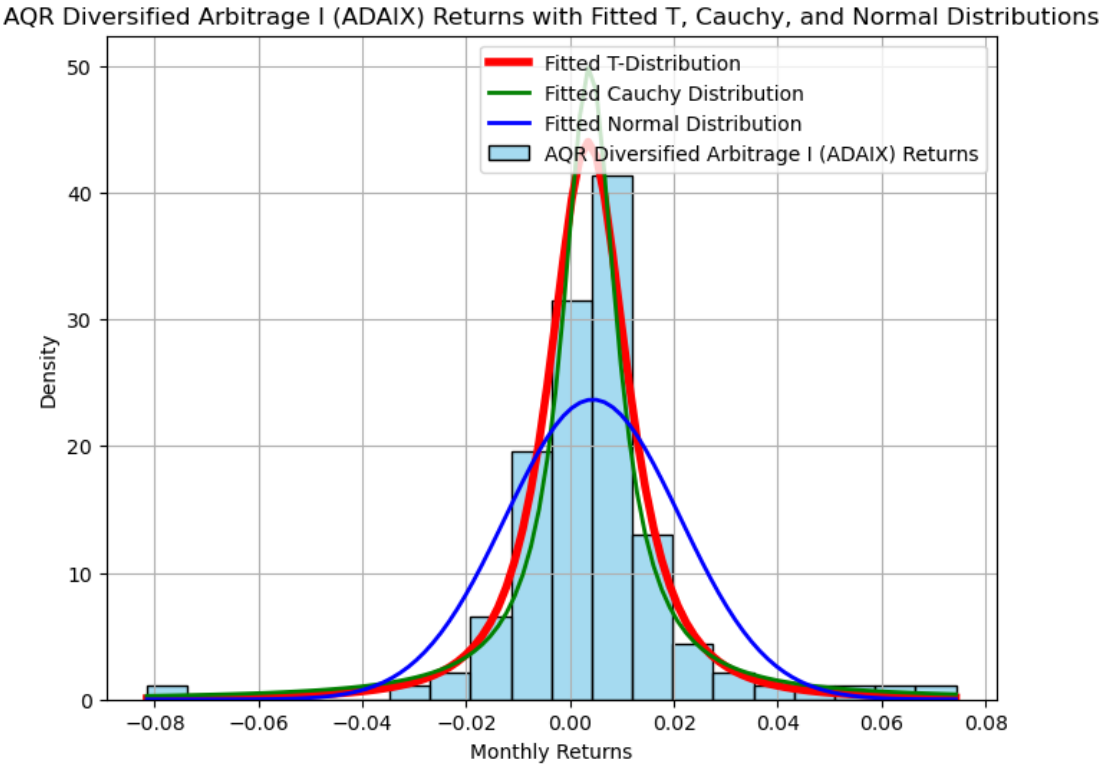
```
        # Save the data needed to recreate the plots
        figures_data.append((asset_returns, x, pdf_t, pdf_cauchy, pdf_normal,␣
 ↪asset))

        # 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)
```

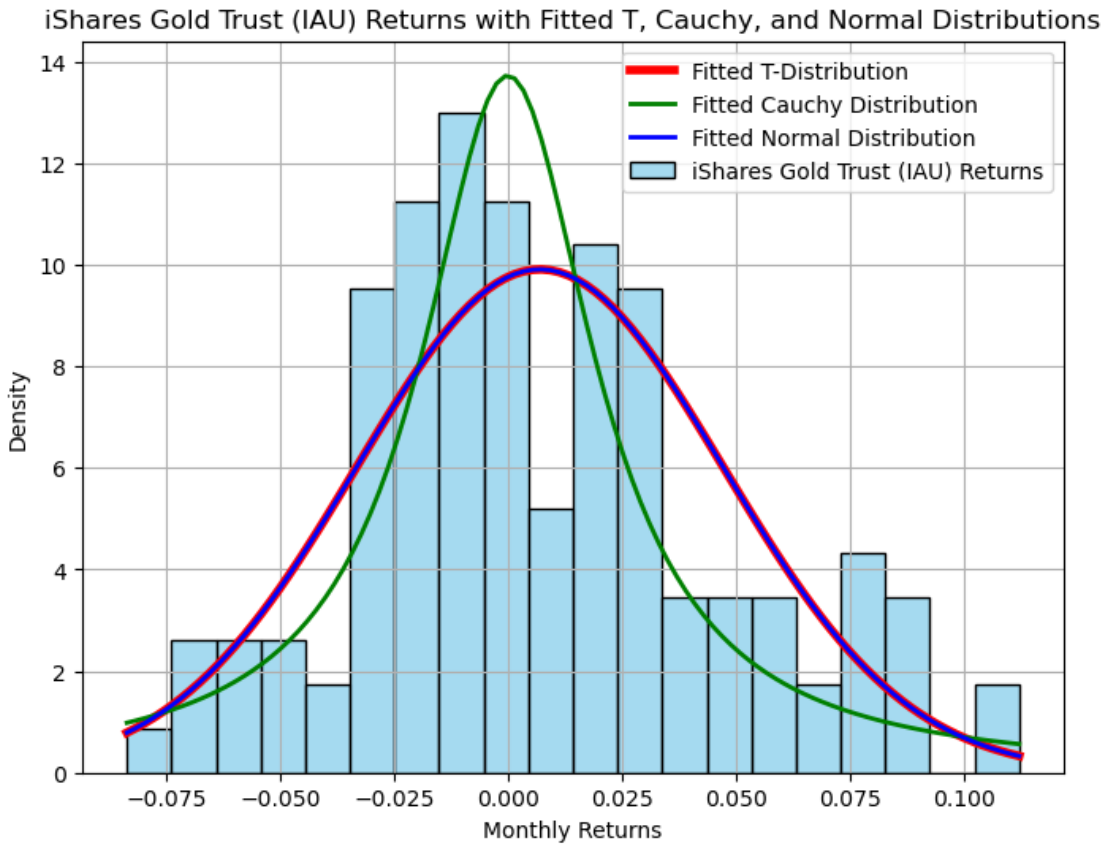
```
    ax.set_xlabel("Monthly Returns", fontsize=8)
    ax.set_ylabel("Density", fontsize=8)
    ax.tick_params(axis='x', rotation=45, labelsize=8)
    ax.tick_params(axis='y', labelsize=8)
    ax.grid(True)
    ax.legend(fontsize=8)

# 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()
```

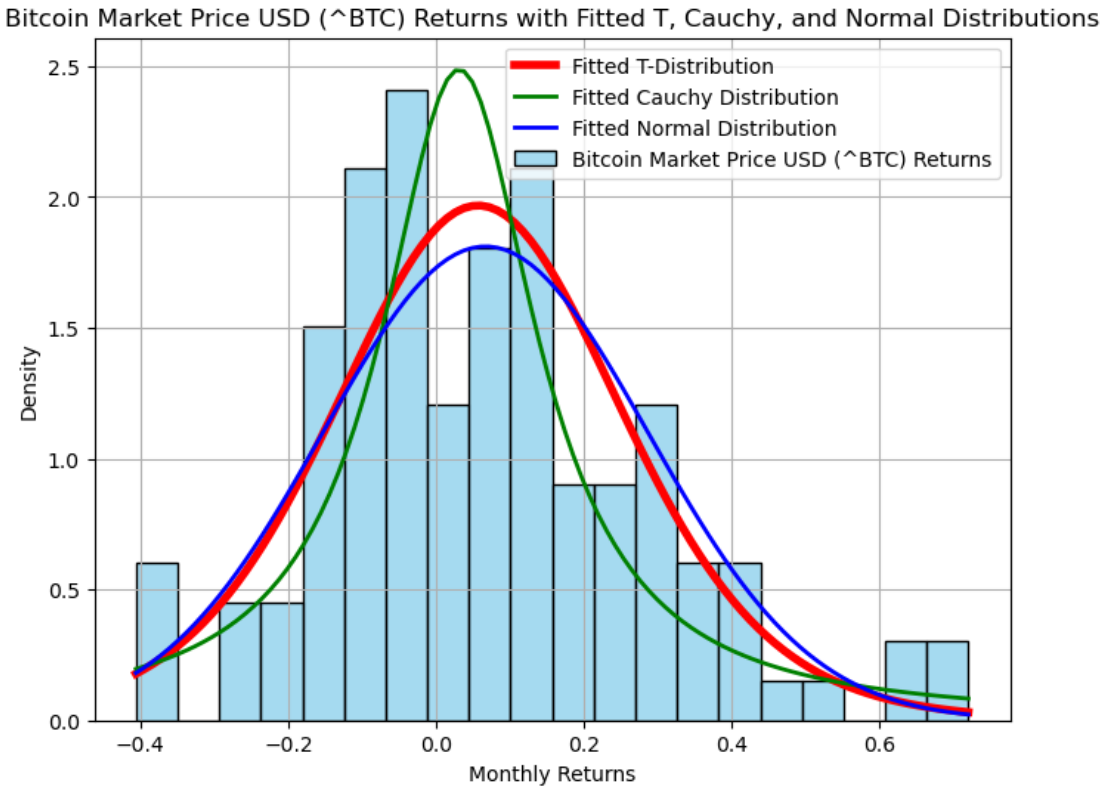


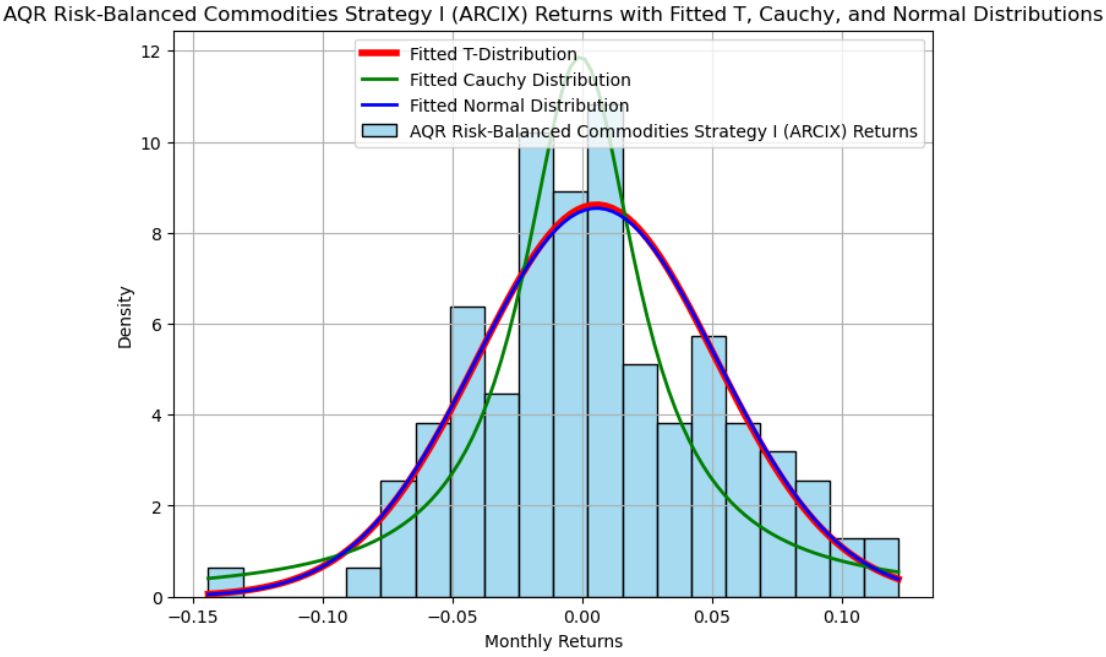

Vanguard Total World Stock ETF (VT) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
Vanguard Total World Stock ETF (VT) Returns
Density
Monthly Returns
0
2
4
6
8
10
12
14
−0.15
−0.10
−0.05
0.00
0.05
0.10

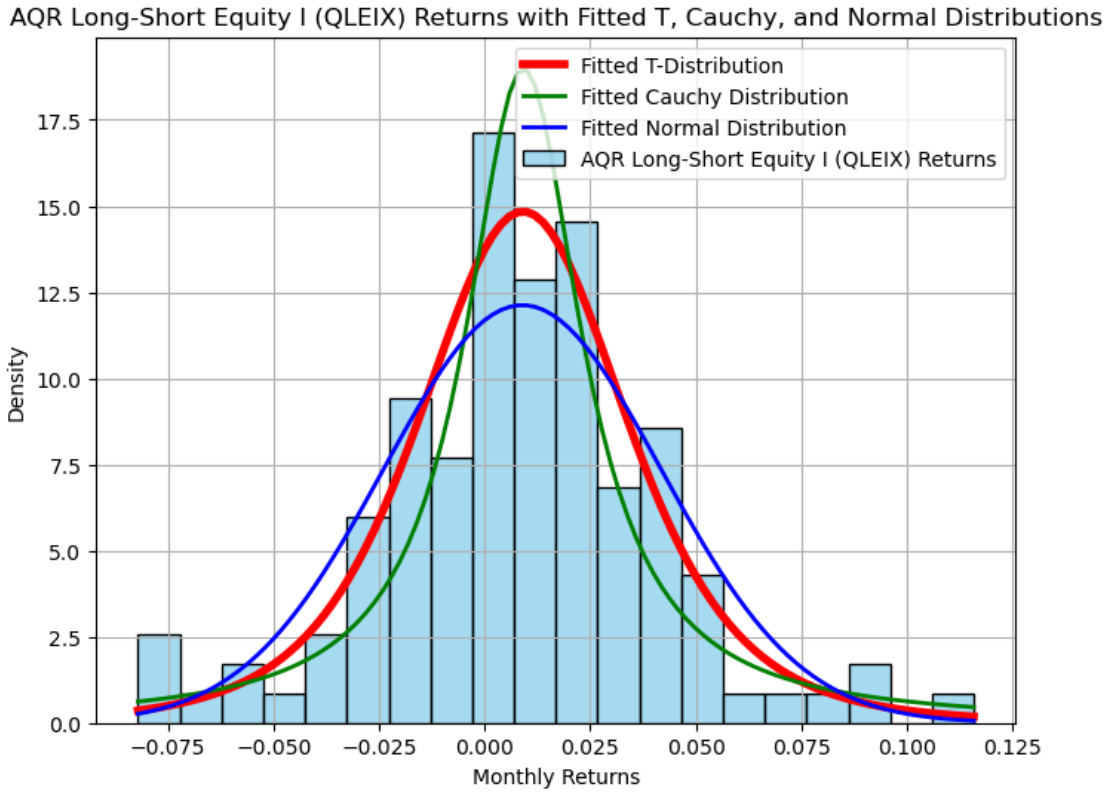

PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ) Returns
Density
Monthly Returns
0
2
4
6
8
−0.10
−0.05
0.00
0.05
0.10
0.15

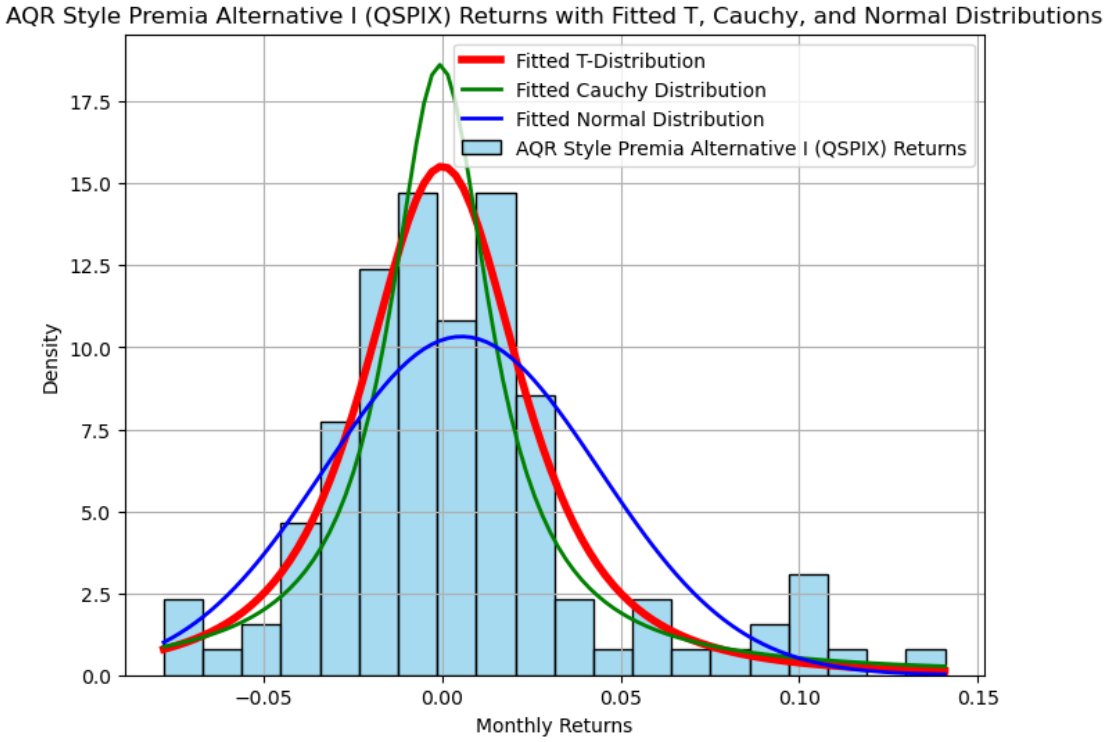

AQR Diversified Arbitrage I (ADAIX) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
AQR Diversified Arbitrage I (ADAIX) Returns
Density
Monthly Returns
0
10
20
30
40
50
−0.08
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06
0.08


iShares Gold Trust (IAU) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
iShares Gold Trust (IAU) Returns
Density
Monthly Returns
0
2
4
6
8
10
12
14
−0.075
−0.050
−0.025
0.000
0.025
0.050
0.075
0.100

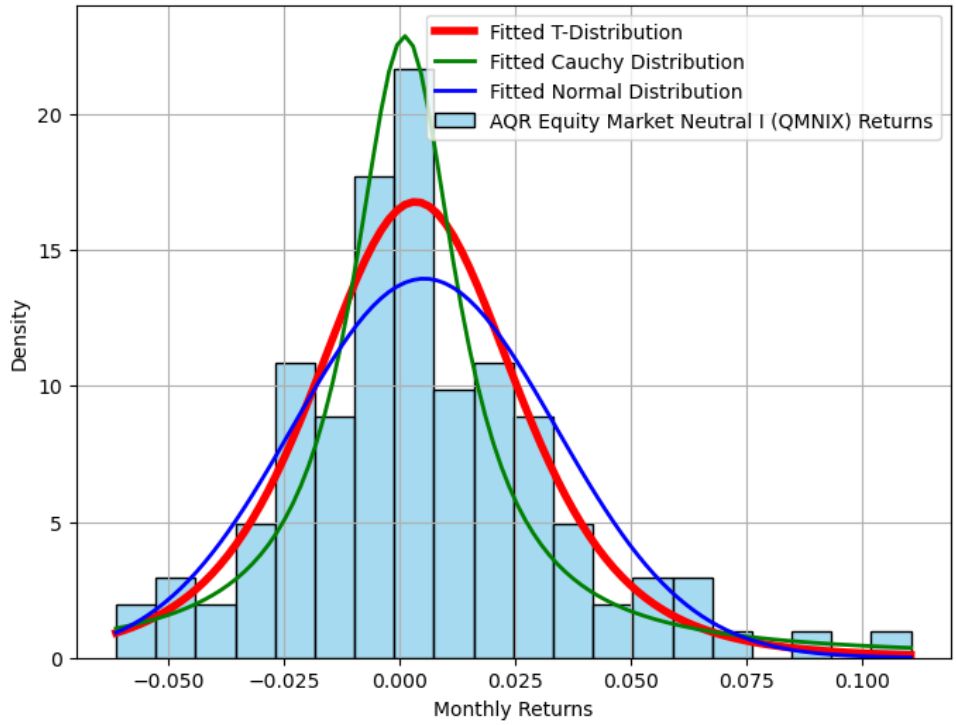

Bitcoin Market Price USD (^BTC) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
Bitcoin Market Price USD (^BTC) Returns
Density
Monthly Returns
2.5
2.0
1.5
1.0
0.5
0.0
−0.4
−0.2
0.0
0.2
0.4
0.6


AQR Risk-Balanced Commodities Strategy I (ARCIX) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
AQR Risk-Balanced Commodities Strategy I (ARCIX) Returns
Density
Monthly Returns
12
10
8
6
4
2
0
−0.15
−0.10
−0.05
0.00
0.05
0.10

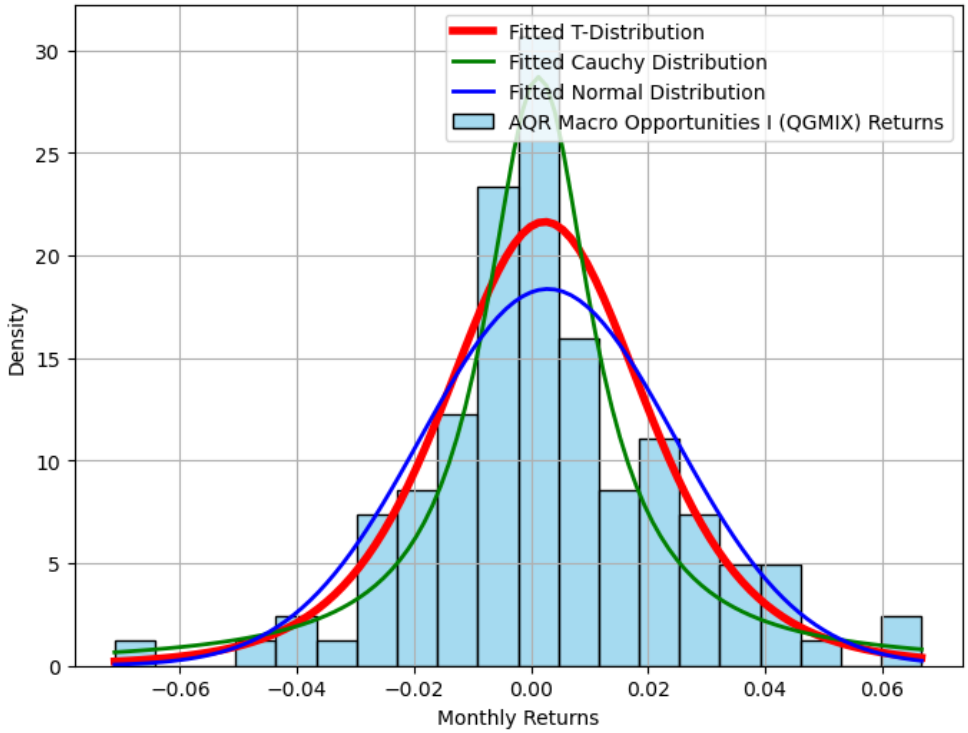

AQR Long-Short Equity I (QLEIX) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
AQR Long-Short Equity I (QLEIX) Returns
Density
Monthly Returns
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
−0.075
−0.050
−0.025
0.000
0.025
0.050
0.075
0.100
0.125


AQR Style Premia Alternative I (QSPIX) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
AQR Style Premia Alternative I (QSPIX) Returns
Density
Monthly Returns
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
−0.05
0.00
0.05
0.10
0.15

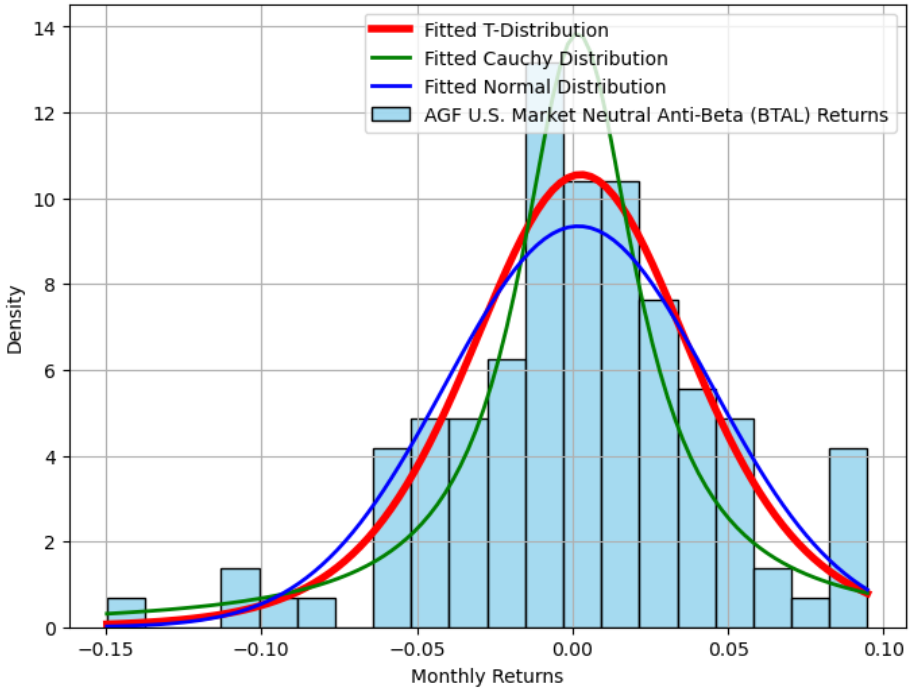
AQR Equity Market Neutral I (QMNIX) Returns with Fitted T, Cauchy, and Normal Distributions



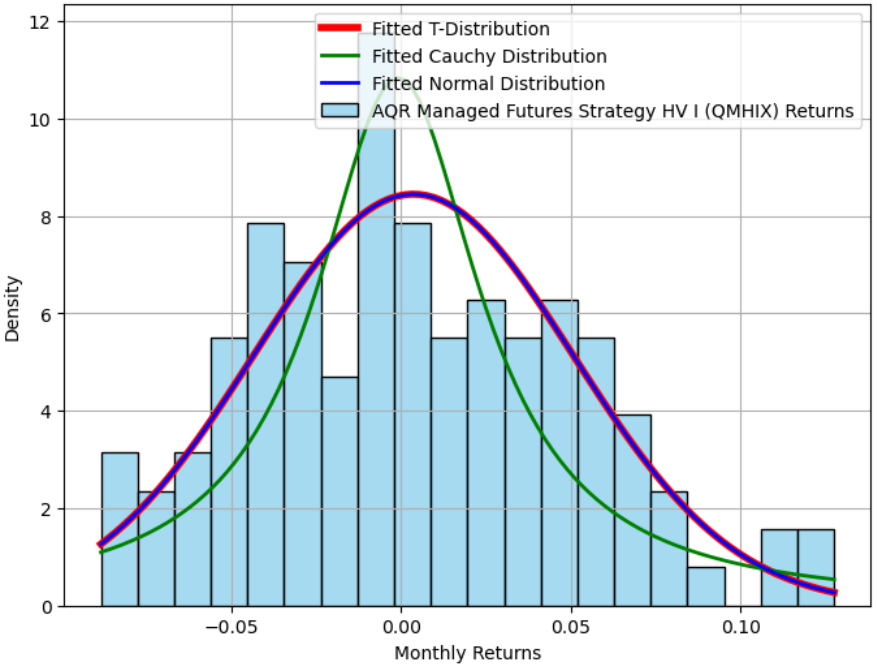
AQR Macro Opportunities I (QGMIX) Returns with Fitted T, Cauchy, and Normal Distributions

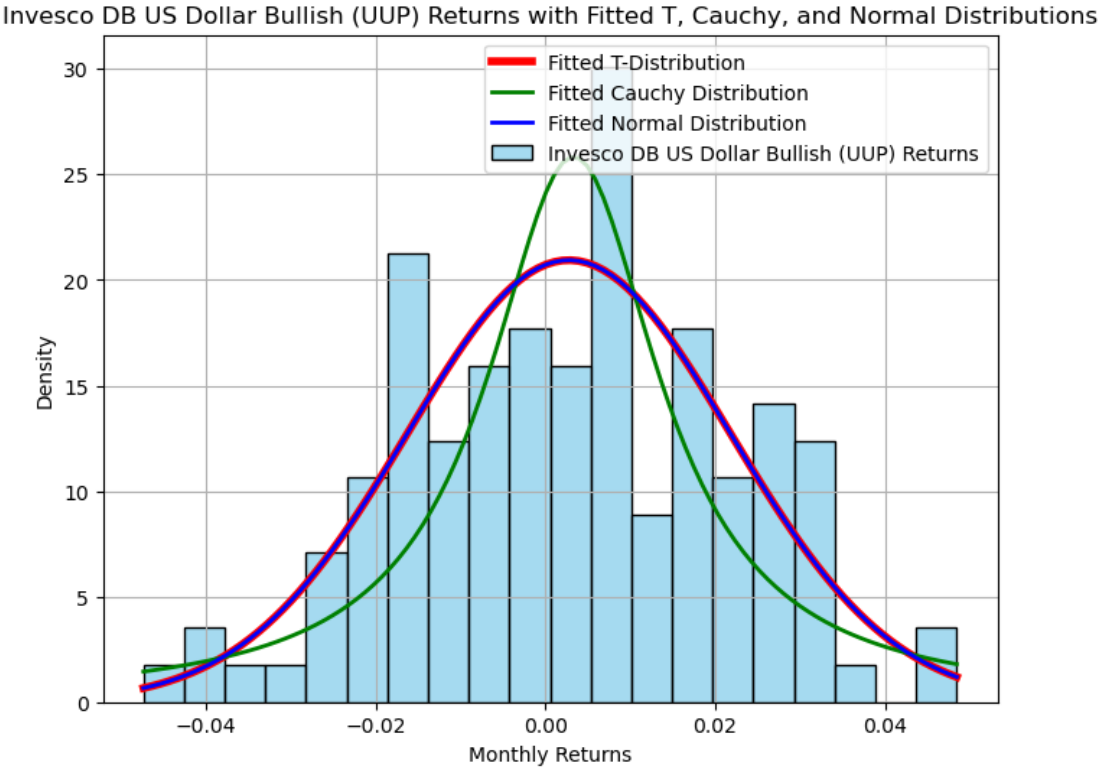


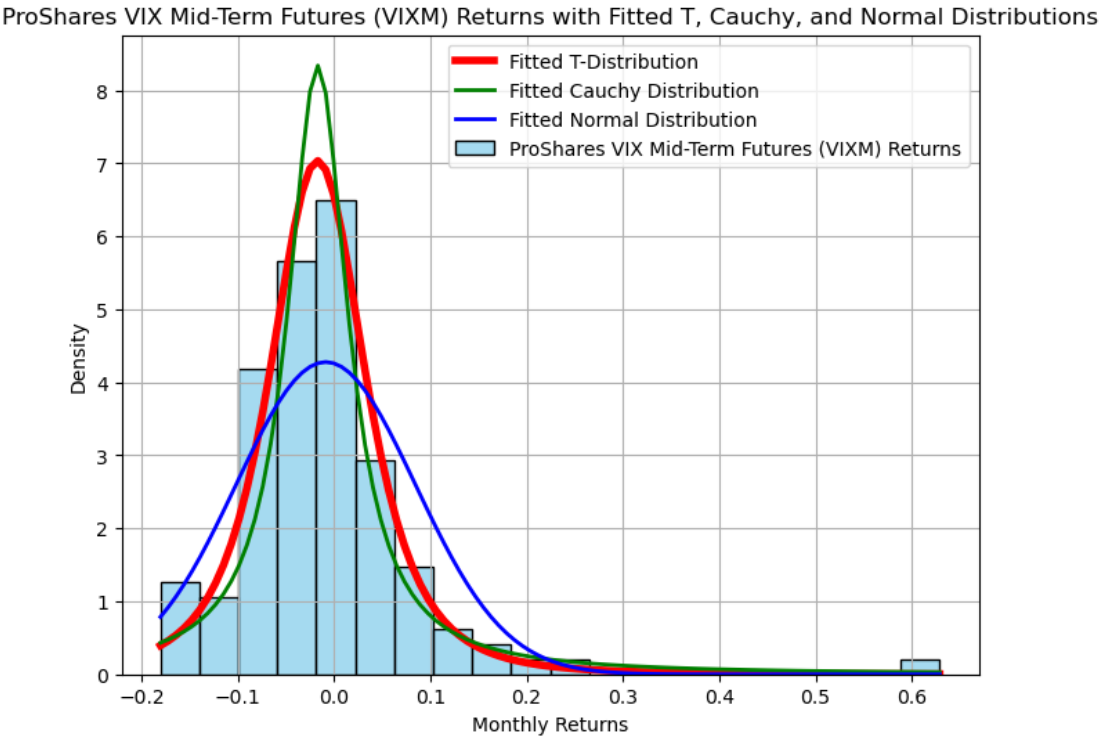
AGF U.S. Market Neutral Anti-Beta (BTAL) Returns with Fitted T, Cauchy, and Normal Distributions

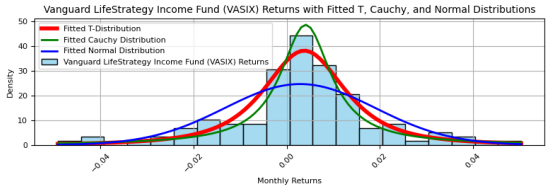


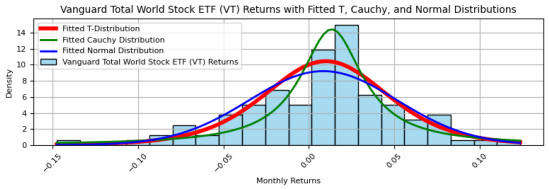
AQR Managed Futures Strategy HV I (QMHIX) Returns with Fitted T, Cauchy, and Normal Distributions

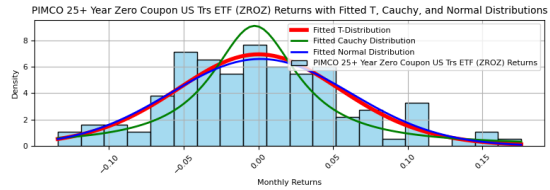


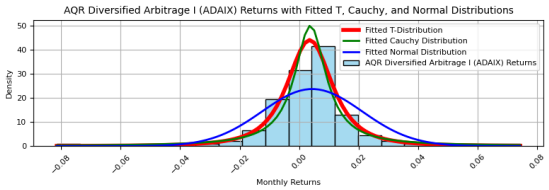

Invesco DB US Dollar Bullish (UUP) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
Invesco DB US Dollar Bullish (UUP) Returns
Density
Monthly Returns
0
5
10
15
20
25
30
−0.04
−0.02
0.00
0.02
0.04

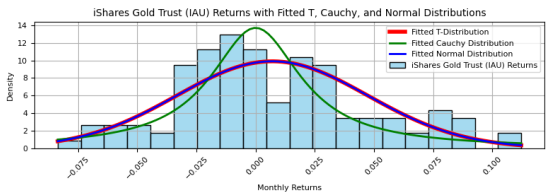

ProShares VIX Mid-Term Futures (VIXM) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
ProShares VIX Mid-Term Futures (VIXM) Returns
Density
Monthly Returns
0
1
2
3
4
5
6
7
8
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
0.5
0.6

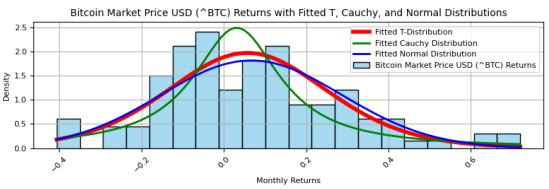

Vanguard LifeStrategy Income Fund (VASIX) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
Vanguard LifeStrategy Income Fund (VASIX) Returns
Density
Monthly Returns


Vanguard Total World Stock ETF (VT) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
Vanguard Total World Stock ETF (VT) Returns
Density
Monthly Returns

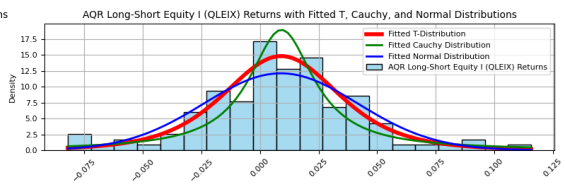

PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ) Returns
Density
Monthly Returns

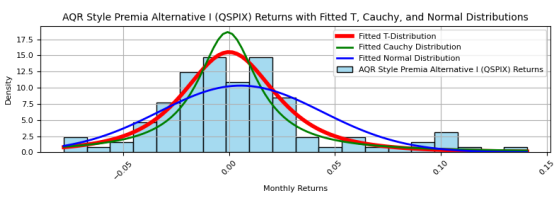

AQR Diversified Arbitrage I (ADAIX) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
AQR Diversified Arbitrage I (ADAIX) Returns
Density
Monthly Returns

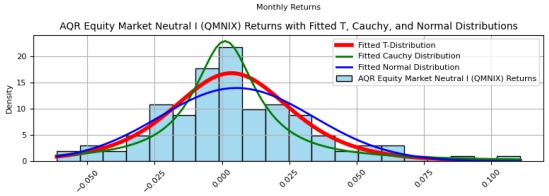

iShares Gold Trust (IAU) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
iShares Gold Trust (IAU) Returns
Density
Monthly Returns

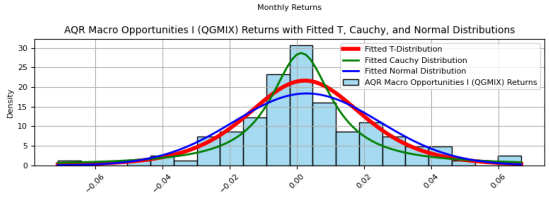

Bitcoin Market Price USD (^BTC) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
Bitcoin Market Price USD (^BTC) Returns
Density
Monthly Returns

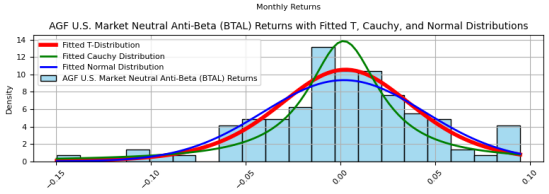

AQR Risk-Balanced Commodities Strategy I (ARCIX) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
AQR Risk-Balanced Commodities Strategy I (ARCIX) Returns
Density
Monthly Returns

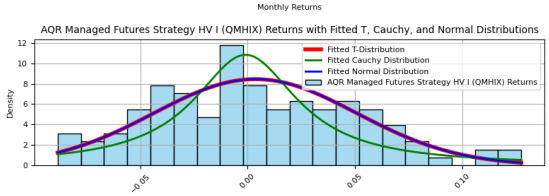

AQR Long-Short Equity I (QLEIX) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
AQR Long-Short Equity I (QLEIX) Returns
Density
Monthly Returns

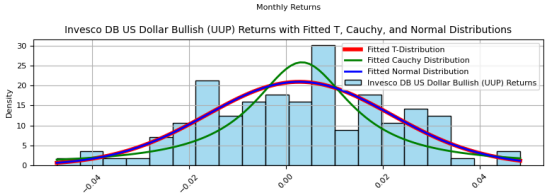

AQR Style Premia Alternative I (QSPIX) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
AQR Style Premia Alternative I (QSPIX) Returns
Density
Monthly Returns

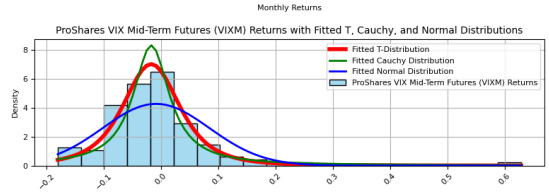

AQR Equity Market Neutral I (QMNIX) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
AQR Equity Market Neutral I (QMNIX) Returns
Density
Monthly Returns


AQR Macro Opportunities I (QGMIX) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
AQR Macro Opportunities I (QGMIX) Returns
Density
Monthly Returns


AGF U.S. Market Neutral Anti-Beta (BTAL) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
AGF U.S. Market Neutral Anti-Beta (BTAL) Returns
Density
Monthly Returns


AQR Managed Futures Strategy HV I (QMHIX) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
AQR Managed Futures Strategy HV I (QMHIX) Returns
Density
Monthly Returns


Invesco DB US Dollar Bullish (UUP) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
Invesco DB US Dollar Bullish (UUP) Returns
Density
Monthly Returns


ProShares VIX Mid-Term Futures (VIXM) Returns with Fitted T, Cauchy, and Normal Distributions
Fitted T-Distribution
Fitted Cauchy Distribution
Fitted Normal Distribution
ProShares VIX Mid-Term Futures (VIXM) Returns
Density
Monthly Returns

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

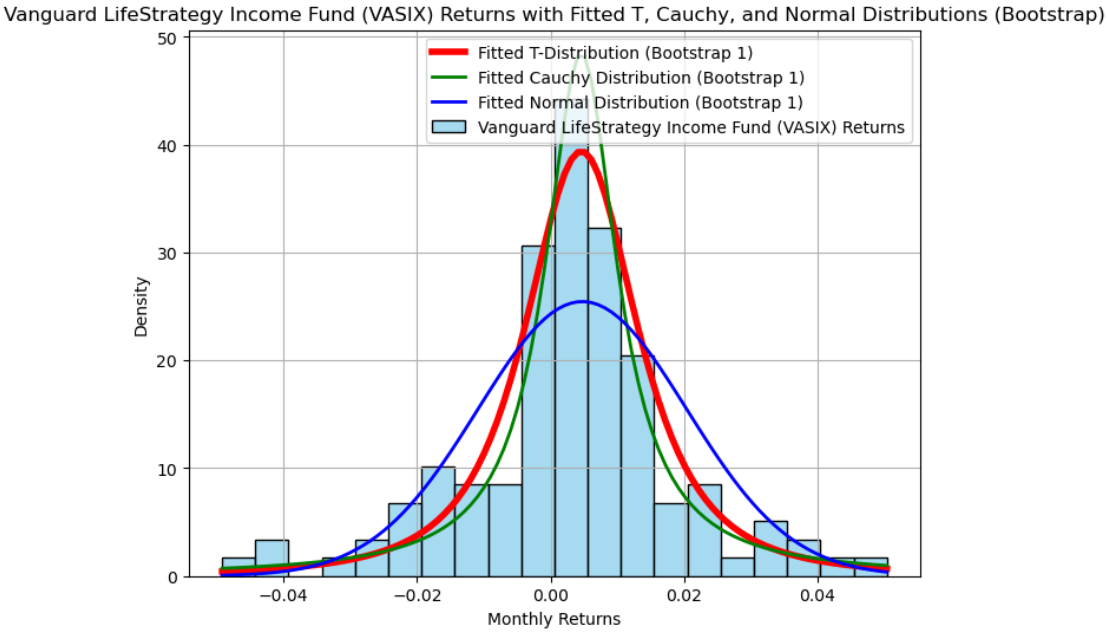
[23]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import t, cauchy, norm
import os

# 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()
```

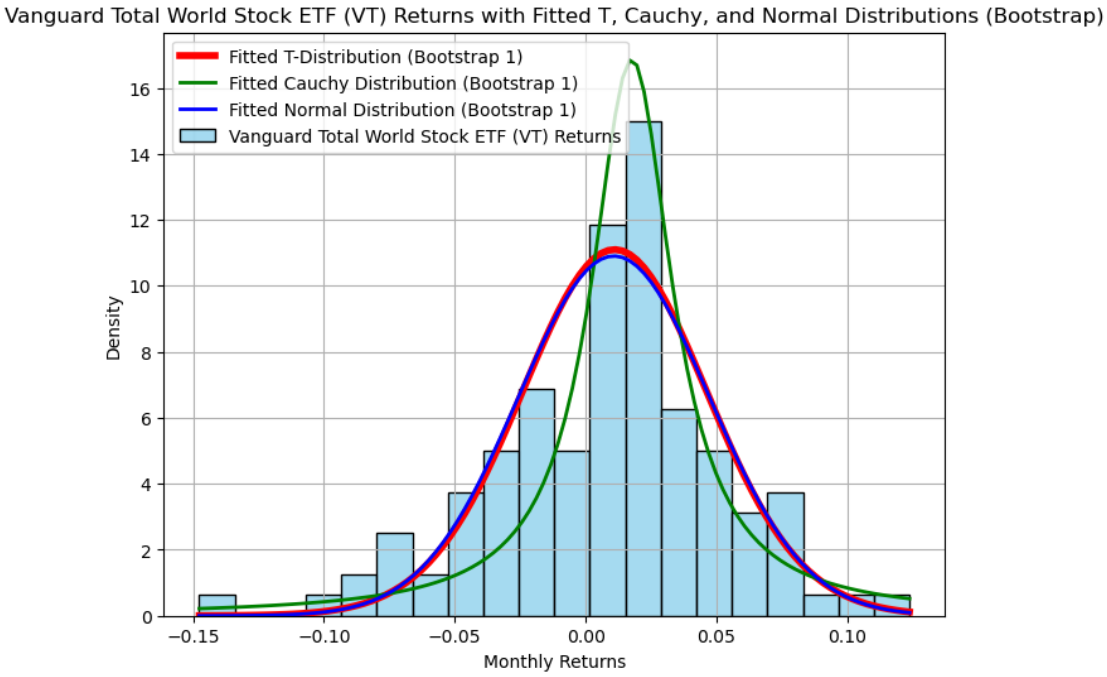
```
# Save the combined figure as a PNG file
combined_fig.savefig(os.path.join(output_folder,␣
 ↪"combined_bootstrap_fitted_distributions.png"), bbox_inches='tight',␣
 ↪dpi=300)  # Save with high resolution

# 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
```

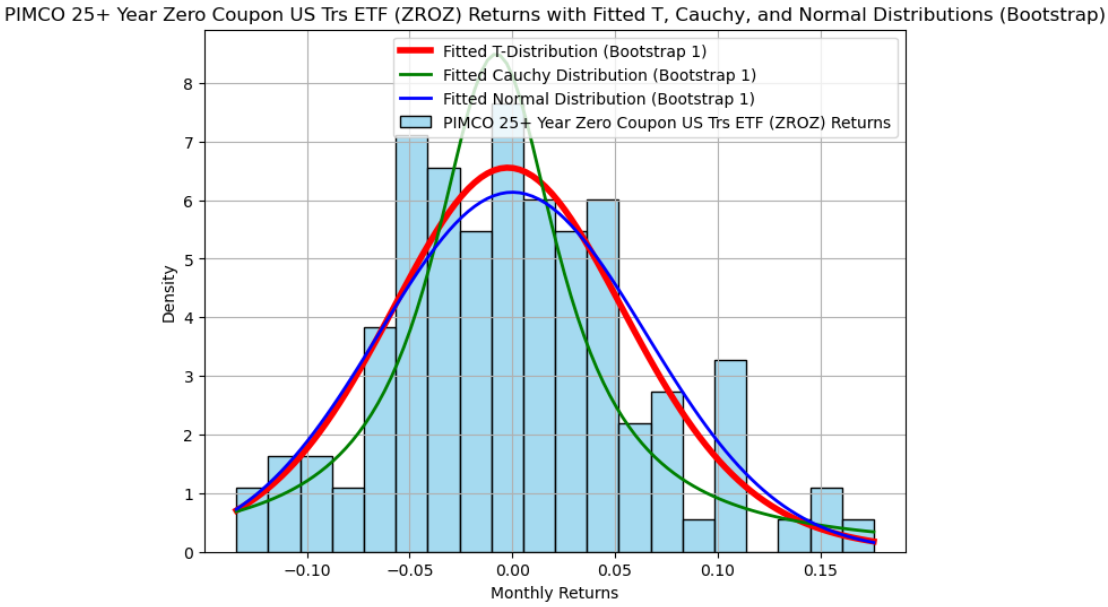



```
Processing asset: Vanguard Total World Stock ETF (VT)
Sample size for Vanguard Total World Stock ETF (VT): 118
Generated 10000 bootstrap samples for Vanguard Total World Stock ETF (VT)
T-distribution (bootstrap sample 1) for Vanguard Total World Stock ETF (VT):
df=42.656054920541166, loc=0.011046908380360803,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

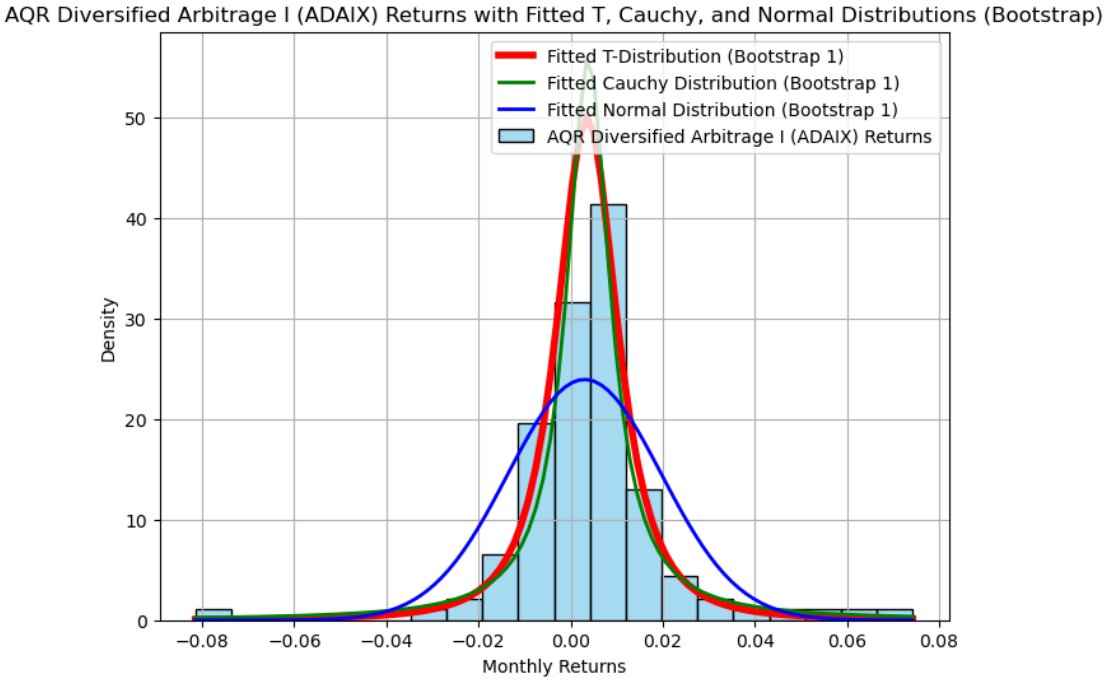
(VT): mu=0.010790677966101694, std=0.036589321836045526



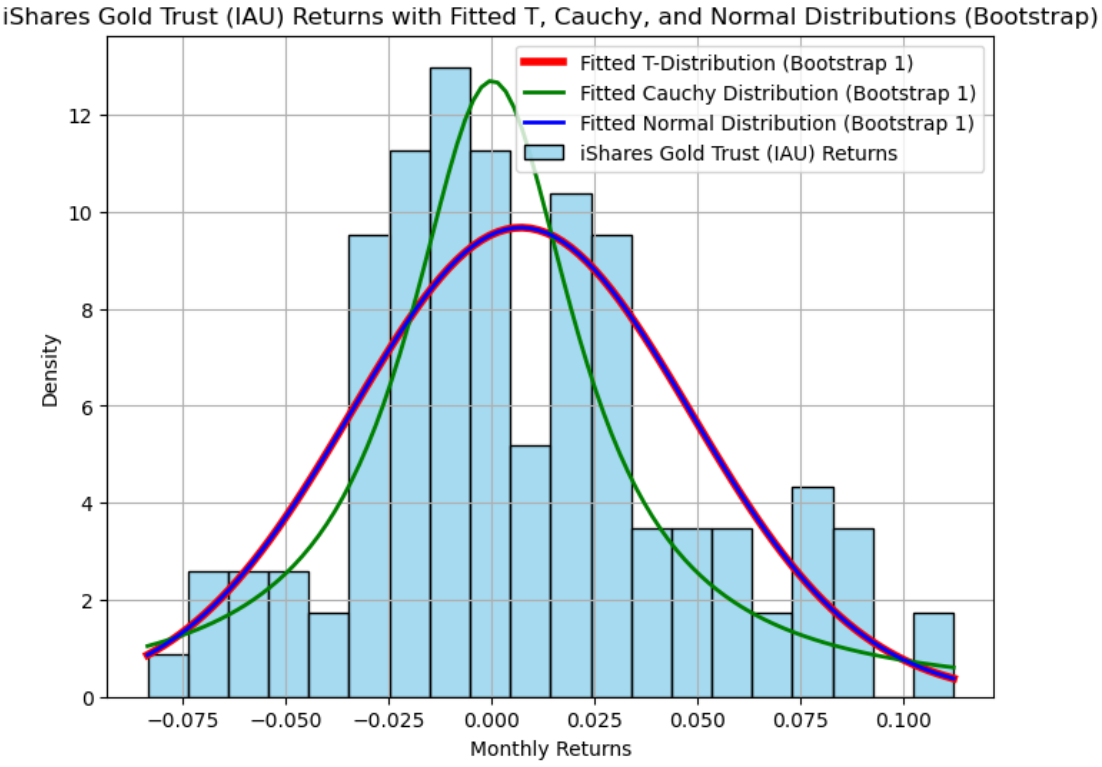
Processing asset: PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)
Sample size for PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ): 118
Generated 10000 bootstrap samples for PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)
T-distribution (bootstrap sample 1) for PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ): df=12.113841337422727, 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



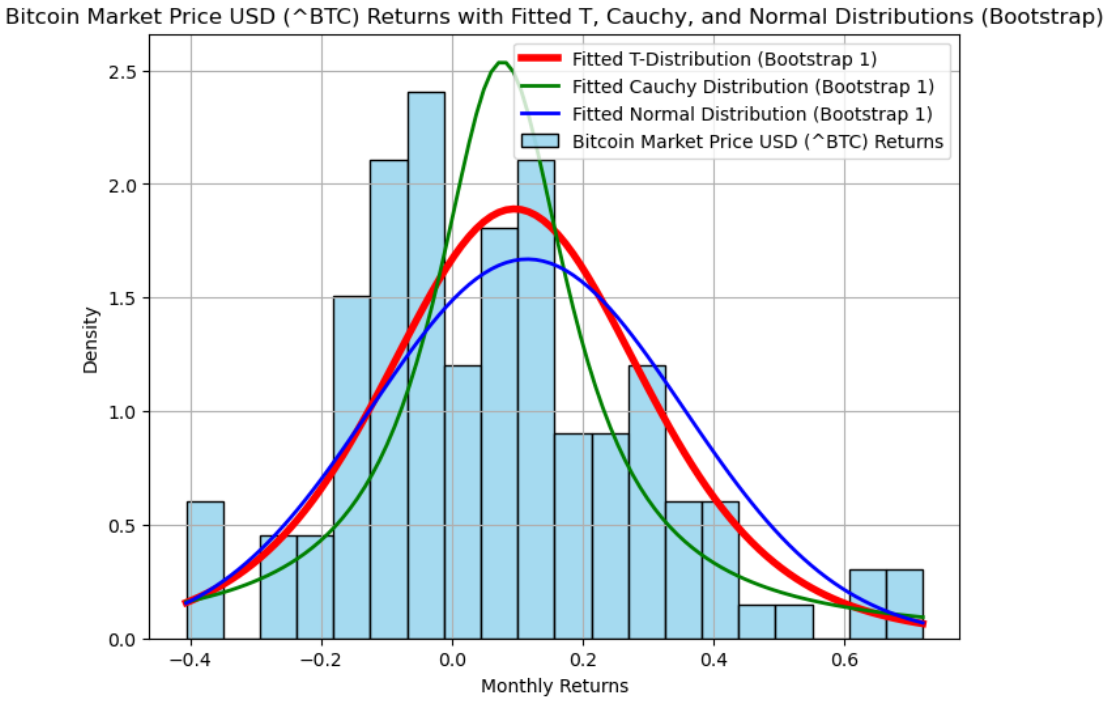

```
Processing asset: AQR Diversified Arbitrage I (ADAIX)
Sample size for AQR Diversified Arbitrage I (ADAIX): 118
Generated 10000 bootstrap samples for AQR Diversified Arbitrage I (ADAIX)
T-distribution (bootstrap sample 1) for AQR Diversified Arbitrage I (ADAIX):
df=2.154571119408492, loc=0.003589952980423613, 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
```



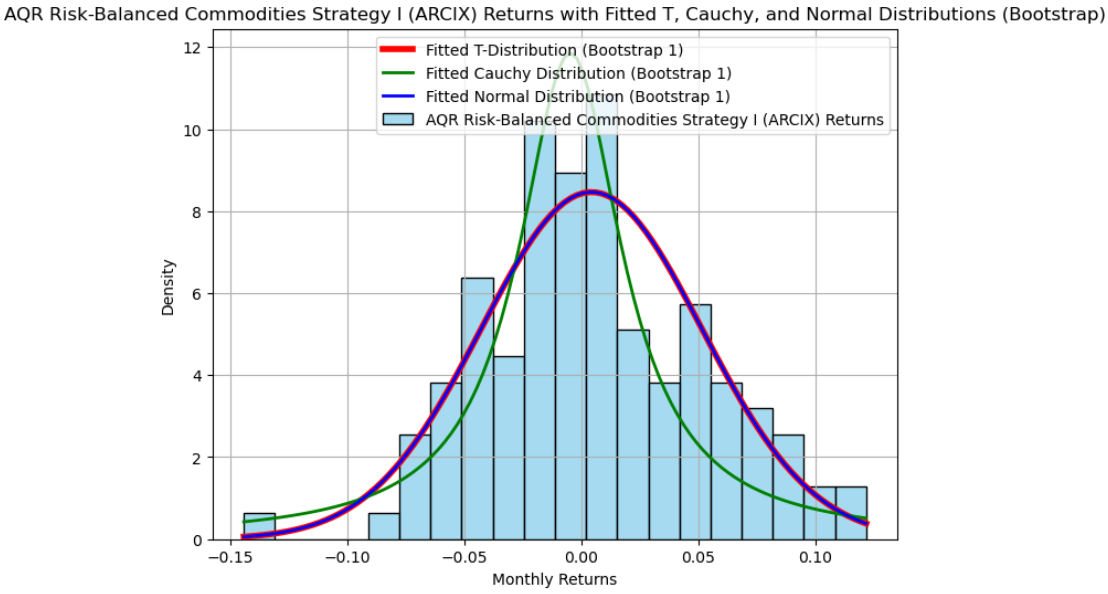

```
Processing asset: iShares Gold Trust (IAU)
Sample size for iShares Gold Trust (IAU): 118
Generated 10000 bootstrap samples for iShares Gold Trust (IAU)
T-distribution (bootstrap sample 1) for iShares Gold Trust (IAU):
df=19448795099.58921, loc=0.007109307847299377, scale=0.041242099198299756
Cauchy 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
```



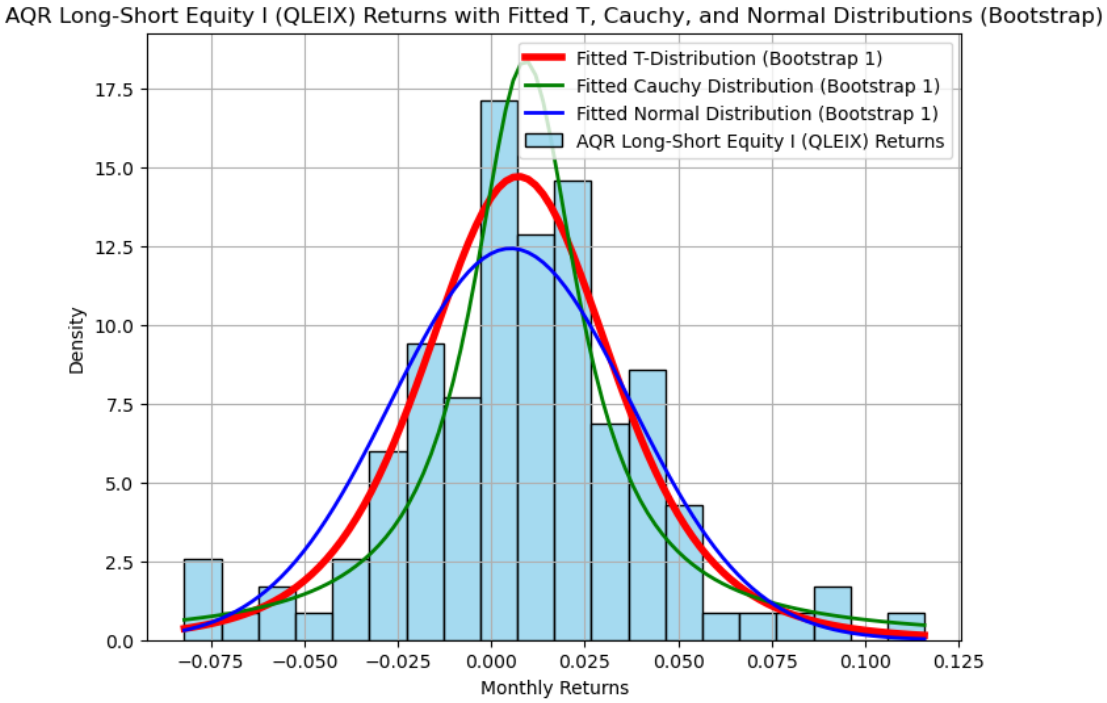

```
Processing asset: Bitcoin Market Price USD (^BTC)
Sample size for Bitcoin Market Price USD (^BTC): 118
Generated 10000 bootstrap samples for Bitcoin Market Price USD (^BTC)
T-distribution (bootstrap sample 1) for Bitcoin Market Price USD (^BTC):
df=6.691724545830047, loc=0.09575876103555714, scale=0.20336266348936424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



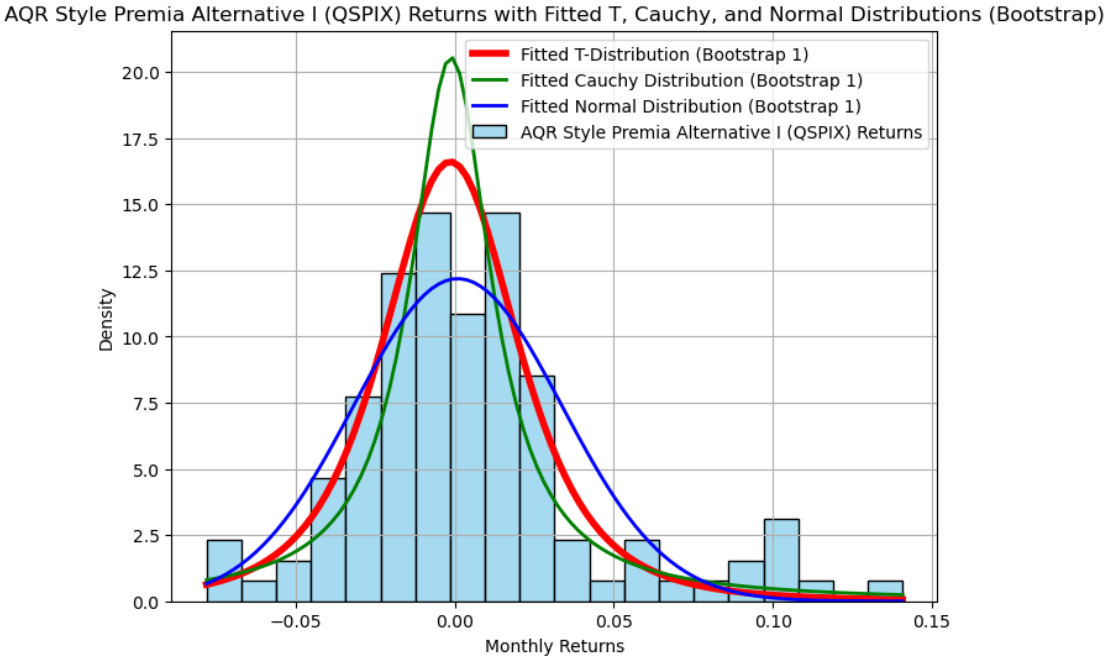

```
Processing asset: AQR Risk-Balanced Commodities Strategy I (ARCIX)
Sample size for AQR Risk-Balanced Commodities Strategy I (ARCIX): 118
Generated 10000 bootstrap samples for AQR Risk-Balanced Commodities Strategy I
(ARCIX)
T-distribution (bootstrap sample 1) for AQR Risk-Balanced Commodities Strategy 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
```



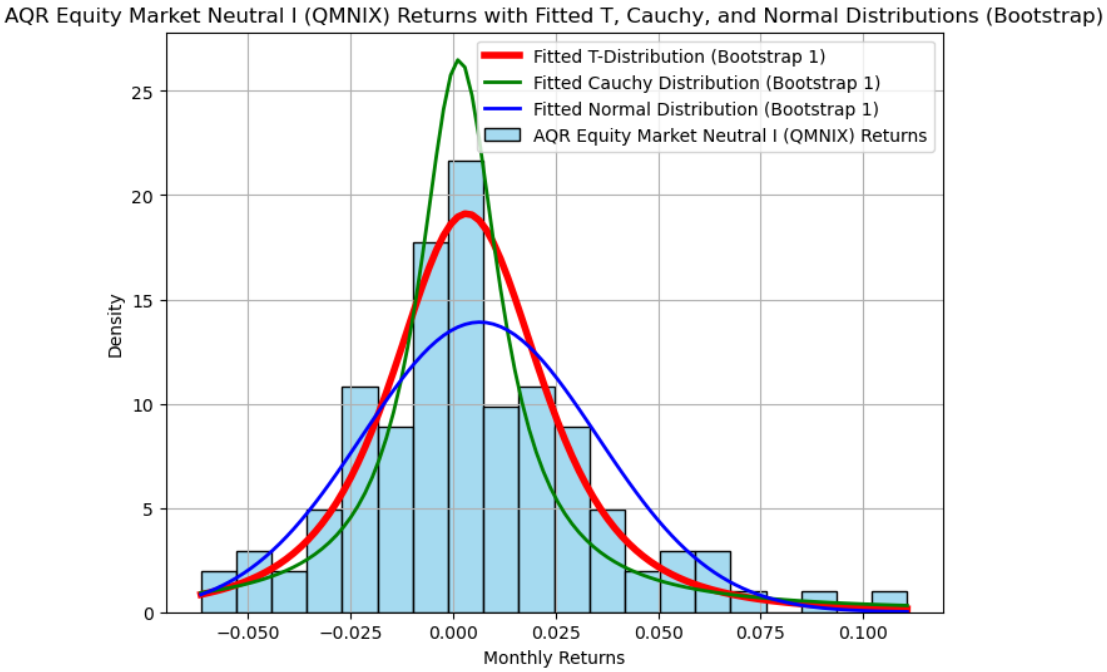

```
Processing asset: AQR Long-Short Equity I (QLEIX)
Sample size for AQR Long-Short Equity I (QLEIX): 118
Generated 10000 bootstrap samples for AQR Long-Short Equity I (QLEIX)
T-distribution (bootstrap sample 1) for AQR Long-Short Equity I (QLEIX):
df=5.252612520881511, loc=0.0072297779725766405, scale=0.025879679335627882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



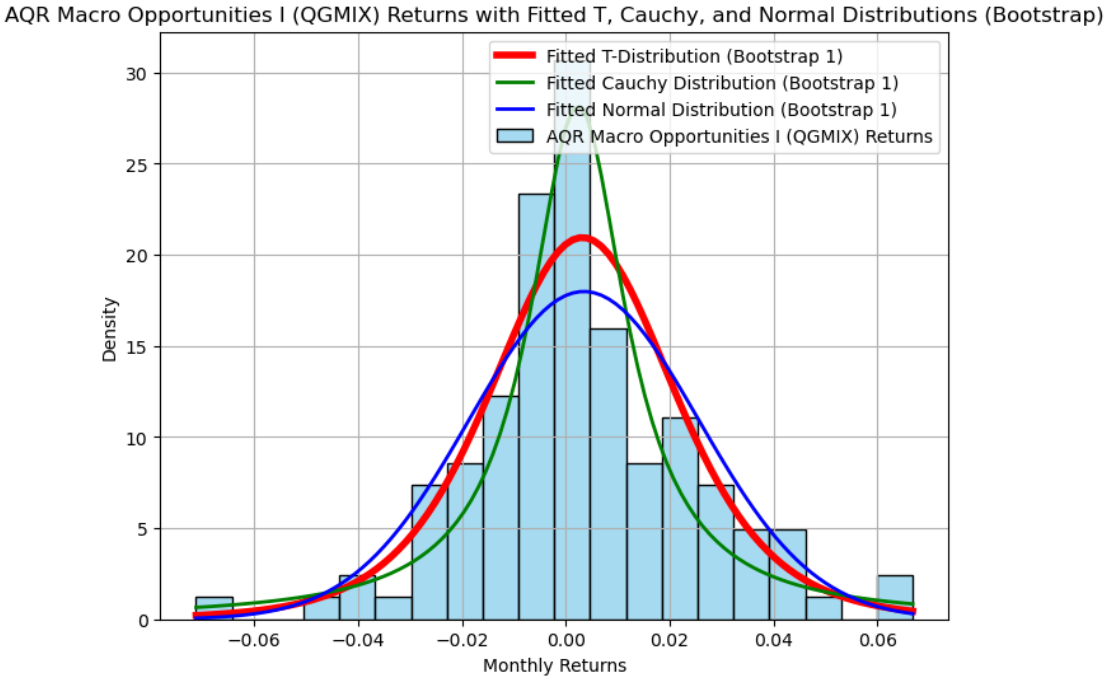

```
Processing asset: AQR Style Premia Alternative I (QSPIX)
Sample size for AQR Style Premia Alternative I (QSPIX): 118
Generated 10000 bootstrap samples for AQR Style Premia Alternative I (QSPIX)
T-distribution (bootstrap sample 1) for AQR Style Premia Alternative I (QSPIX):
df=3.288436286905494, loc=-0.001418549077672707, 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
```



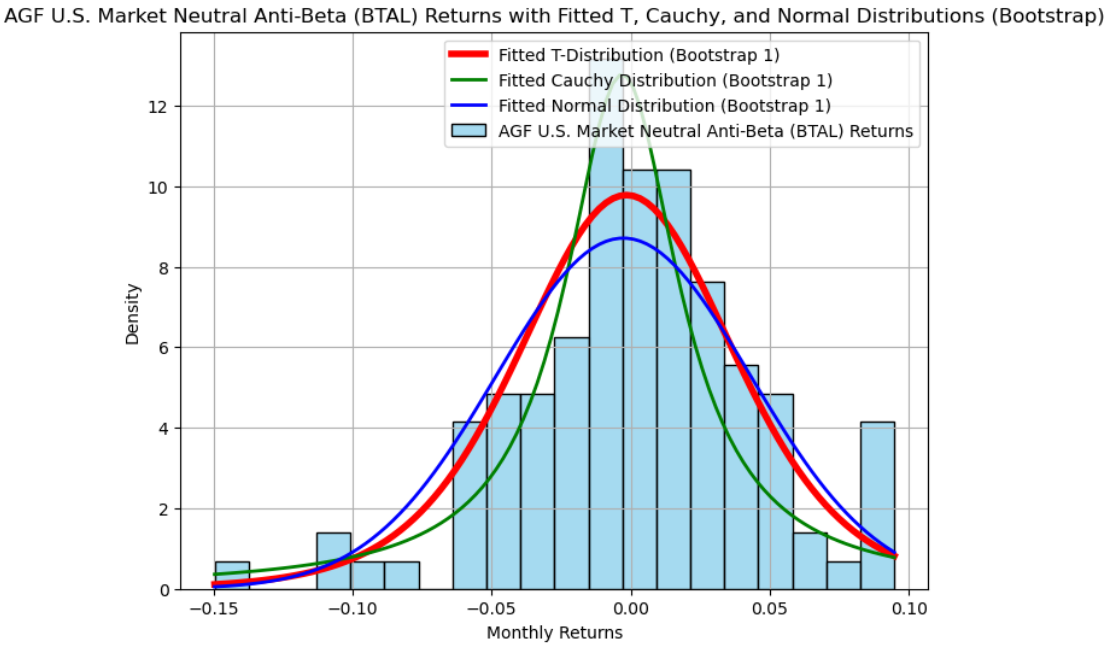

```
Processing asset: AQR Equity Market Neutral I (QMNIX)
Sample size for AQR Equity Market Neutral I (QMNIX): 118
Generated 10000 bootstrap samples for AQR Equity Market Neutral I (QMNIX)
T-distribution (bootstrap sample 1) for AQR Equity Market Neutral I (QMNIX):
df=2.9402221310381007, loc=0.003331117160879359, 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
```



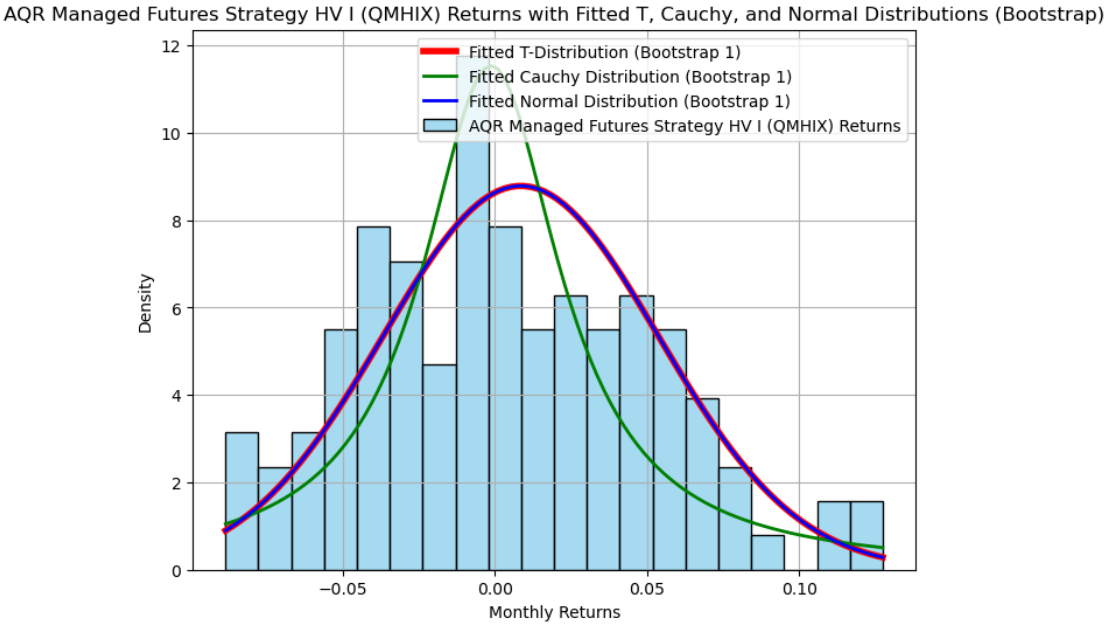

```
Processing asset: AQR Macro Opportunities I (QGMIX)
Sample size for AQR Macro Opportunities I (QGMIX): 118
Generated 10000 bootstrap samples for AQR Macro Opportunities I (QGMIX)
T-distribution (bootstrap sample 1) for AQR Macro Opportunities I (QGMIX):
df=5.854242147327145, loc=0.0031937358652952145, 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
```



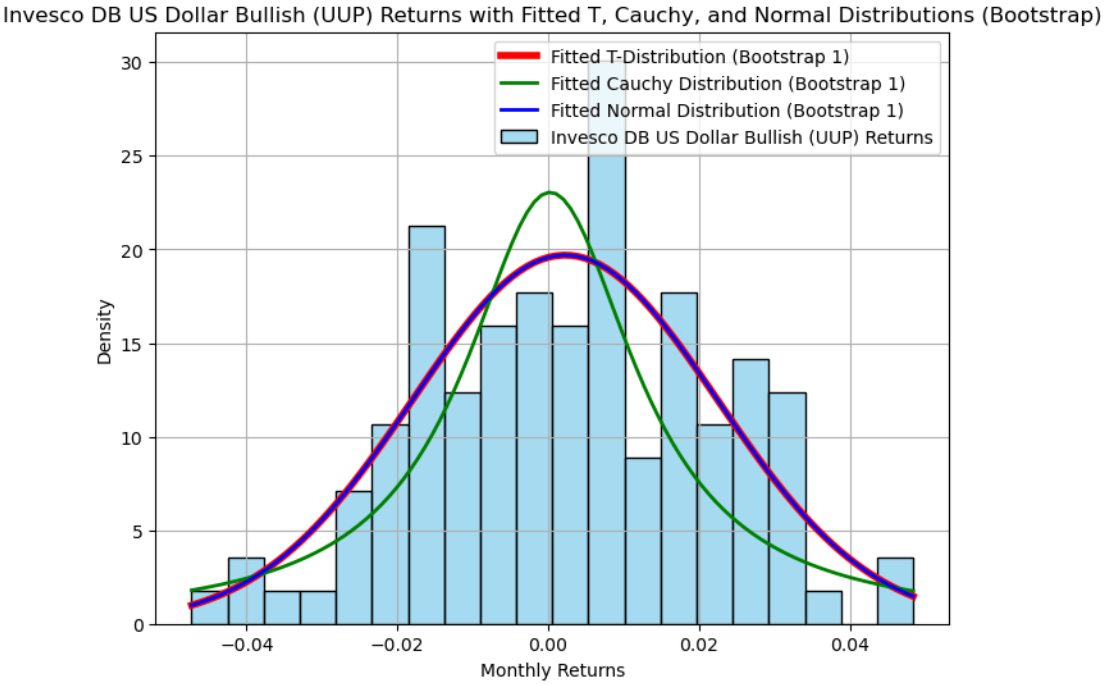

```
Processing asset: AGF U.S. Market Neutral Anti-Beta (BTAL)
Sample size for AGF U.S. Market Neutral Anti-Beta (BTAL): 118
Generated 10000 bootstrap samples for AGF U.S. Market Neutral Anti-Beta (BTAL)
T-distribution (bootstrap sample 1) for AGF U.S. Market Neutral Anti-Beta
(BTAL): df=7.5244837144737815, 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
```



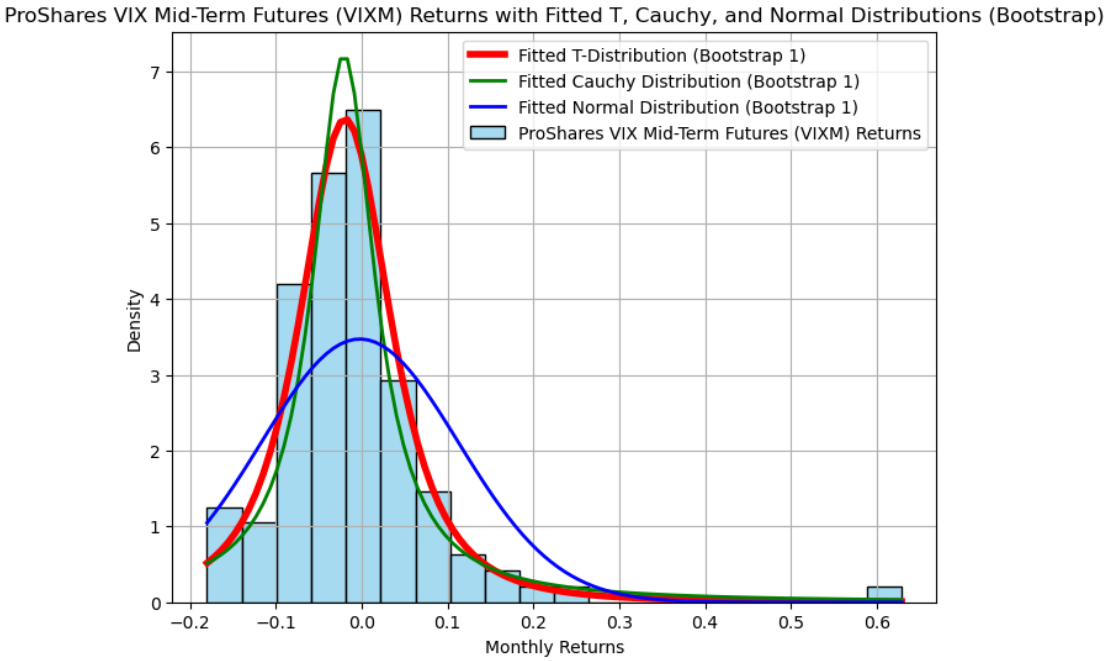

```
Processing asset: AQR Managed Futures Strategy HV I (QMHIX)
Sample size for AQR Managed Futures Strategy HV I (QMHIX): 118
Generated 10000 bootstrap samples for AQR Managed Futures Strategy HV I (QMHIX)
T-distribution (bootstrap sample 1) for AQR Managed Futures Strategy HV I
(QMHIX): df=215425607.84337816, 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
```

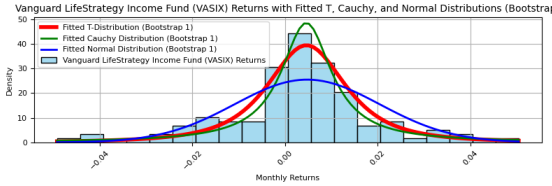


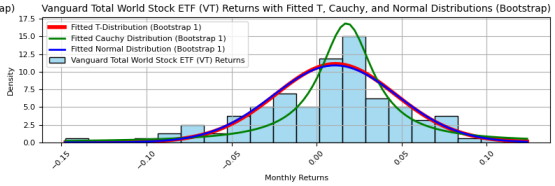

```
Processing asset: Invesco DB US Dollar Bullish (UUP)
Sample size for Invesco DB US Dollar Bullish (UUP): 118
Generated 10000 bootstrap samples for Invesco DB US Dollar Bullish (UUP)
T-distribution (bootstrap sample 1) for Invesco DB US Dollar Bullish (UUP):
df=3170640257.430148, loc=0.0022296703801621444, 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
```

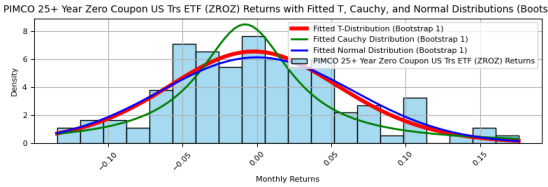


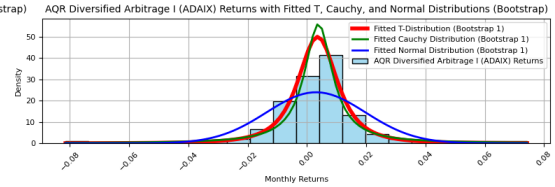

```
Processing asset: ProShares VIX Mid-Term Futures (VIXM)
Sample size for ProShares VIX Mid-Term Futures (VIXM): 118
Generated 10000 bootstrap samples for ProShares VIX Mid-Term Futures (VIXM)
T-distribution (bootstrap sample 1) for ProShares VIX Mid-Term Futures (VIXM):
df=2.4489111968240422, loc=-0.019283755577882818, 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
```

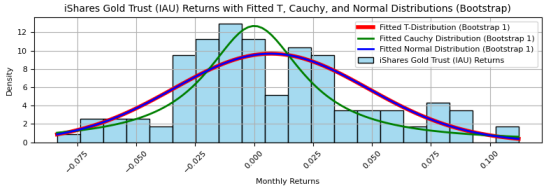

ProShares VIX Mid-Term Futures (VIXM) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
Fitted T-Distribution (Bootstrap 1)
Fitted Cauchy Distribution (Bootstrap 1)
Fitted Normal Distribution (Bootstrap 1)
ProShares VIX Mid-Term Futures (VIXM) Returns
Density
Monthly Returns
0
1
2
3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

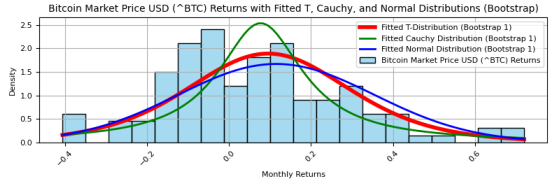

Vanguard LifeStrategy Income Fund (VASIX) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
Fitted T-Distribution (Bootstrap 1)
Fitted Cauchy Distribution (Bootstrap 1)
Fitted Normal Distribution (Bootstrap 1)
Vanguard LifeStrategy Income Fund (VASIX) Returns
Density
Monthly Returns

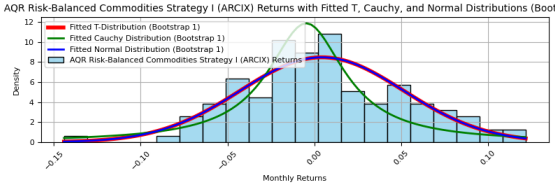

Vanguard Total World Stock ETF (VT) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
Fitted T-Distribution (Bootstrap 1)
Fitted Cauchy Distribution (Bootstrap 1)
Fitted Normal Distribution (Bootstrap 1)
Vanguard Total World Stock ETF (VT) Returns
Density
Monthly Returns

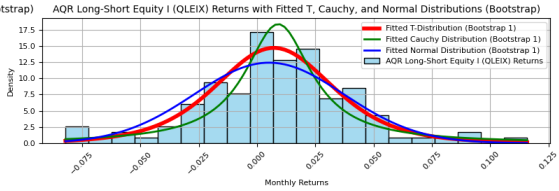

PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
Fitted T-Distribution (Bootstrap 1)
Fitted Cauchy Distribution (Bootstrap 1)
Fitted Normal Distribution (Bootstrap 1)
PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ) Returns
Density
Monthly Returns

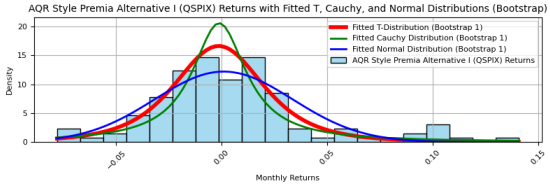

AQR Diversified Arbitrage I (ADAIX) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
Fitted T-Distribution (Bootstrap 1)
Fitted Cauchy Distribution (Bootstrap 1)
Fitted Normal Distribution (Bootstrap 1)
AQR Diversified Arbitrage I (ADAIX) Returns
Density
Monthly Returns

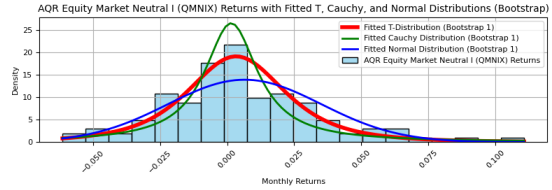

iShares Gold Trust (IAU) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
Fitted T-Distribution (Bootstrap 1)
Fitted Cauchy Distribution (Bootstrap 1)
Fitted Normal Distribution (Bootstrap 1)
iShares Gold Trust (IAU) Returns
Density
Monthly Returns

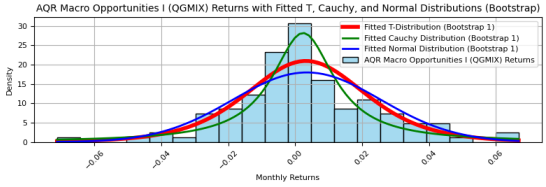

Bitcoin Market Price USD (^BTC) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
Fitted T-Distribution (Bootstrap 1)
Fitted Cauchy Distribution (Bootstrap 1)
Fitted Normal Distribution (Bootstrap 1)
Bitcoin Market Price USD (^BTC) Returns
Density
Monthly Returns

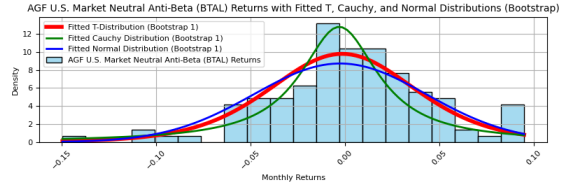

AQR Risk-Balanced Commodities Strategy I (ARCIX) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
Fitted T-Distribution (Bootstrap 1)
Fitted Cauchy Distribution (Bootstrap 1)
Fitted Normal Distribution (Bootstrap 1)
AQR Risk-Balanced Commodities Strategy I (ARCIX) Returns
Density
Monthly Returns

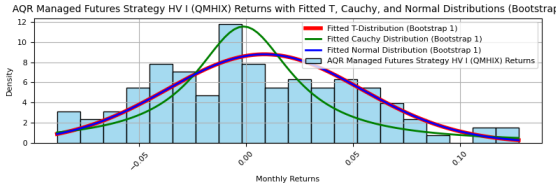

AQR Long-Short Equity I (QLEIX) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
Fitted T-Distribution (Bootstrap 1)
Fitted Cauchy Distribution (Bootstrap 1)
Fitted Normal Distribution (Bootstrap 1)
AQR Long-Short Equity I (QLEIX) Returns
Density
Monthly Returns

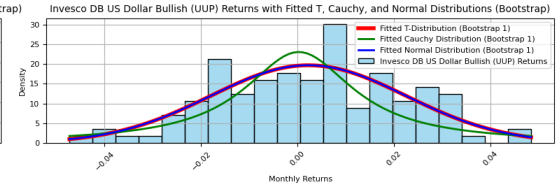

AQR Style Premia Alternative I (QSPIX) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
Fitted T-Distribution (Bootstrap 1)
Fitted Cauchy Distribution (Bootstrap 1)
Fitted Normal Distribution (Bootstrap 1)
AQR Style Premia Alternative I (QSPIX) Returns
Density
Monthly Returns

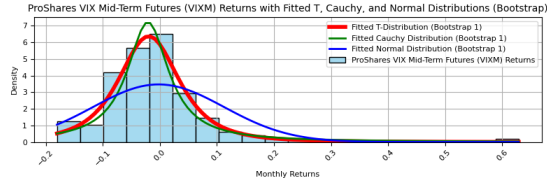

AQR Equity Market Neutral I (QMNIX) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
Fitted T-Distribution (Bootstrap 1)
Fitted Cauchy Distribution (Bootstrap 1)
Fitted Normal Distribution (Bootstrap 1)
AQR Equity Market Neutral I (QMNIX) Returns
Density
Monthly Returns


AQR Macro Opportunities I (QGMIX) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
Fitted T-Distribution (Bootstrap 1)
Fitted Cauchy Distribution (Bootstrap 1)
Fitted Normal Distribution (Bootstrap 1)
AQR Macro Opportunities I (QGMIX) Returns
Density
Monthly Returns


AGF U.S. Market Neutral Anti-Beta (BTAL) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
Fitted T-Distribution (Bootstrap 1)
Fitted Cauchy Distribution (Bootstrap 1)
Fitted Normal Distribution (Bootstrap 1)
AGF U.S. Market Neutral Anti-Beta (BTAL) Returns
Density
Monthly Returns


AQR Managed Futures Strategy HV I (QMHIX) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
Fitted T-Distribution (Bootstrap 1)
Fitted Cauchy Distribution (Bootstrap 1)
Fitted Normal Distribution (Bootstrap 1)
AQR Managed Futures Strategy HV I (QMHIX) Returns
Density
Monthly Returns

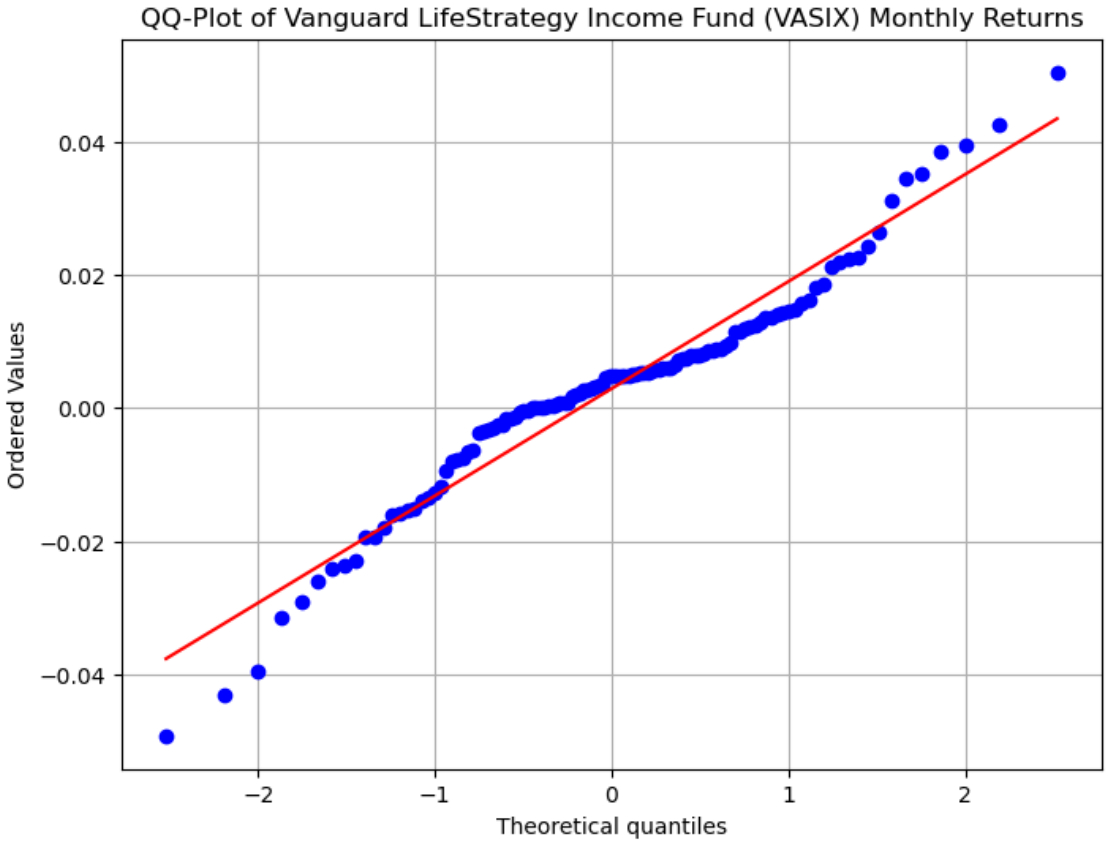

Invesco DB US Dollar Bullish (UUP) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
Fitted T-Distribution (Bootstrap 1)
Fitted Cauchy Distribution (Bootstrap 1)
Fitted Normal Distribution (Bootstrap 1)
Invesco DB US Dollar Bullish (UUP) Returns
Density
Monthly Returns

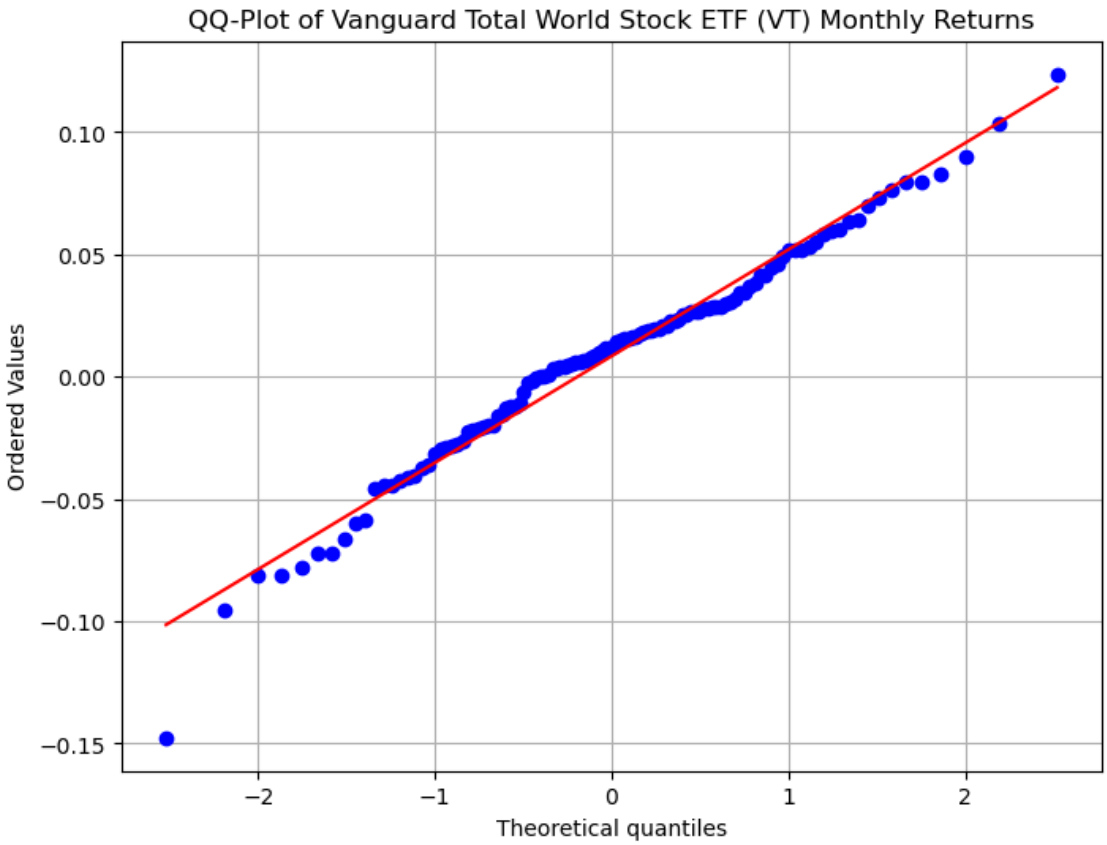

ProShares VIX Mid-Term Futures (VIXM) Returns with Fitted T, Cauchy, and Normal Distributions (Bootstrap)
Fitted T-Distribution (Bootstrap 1)
Fitted Cauchy Distribution (Bootstrap 1)
Fitted Normal Distribution (Bootstrap 1)
ProShares VIX Mid-Term Futures (VIXM) Returns
Density
Monthly Returns

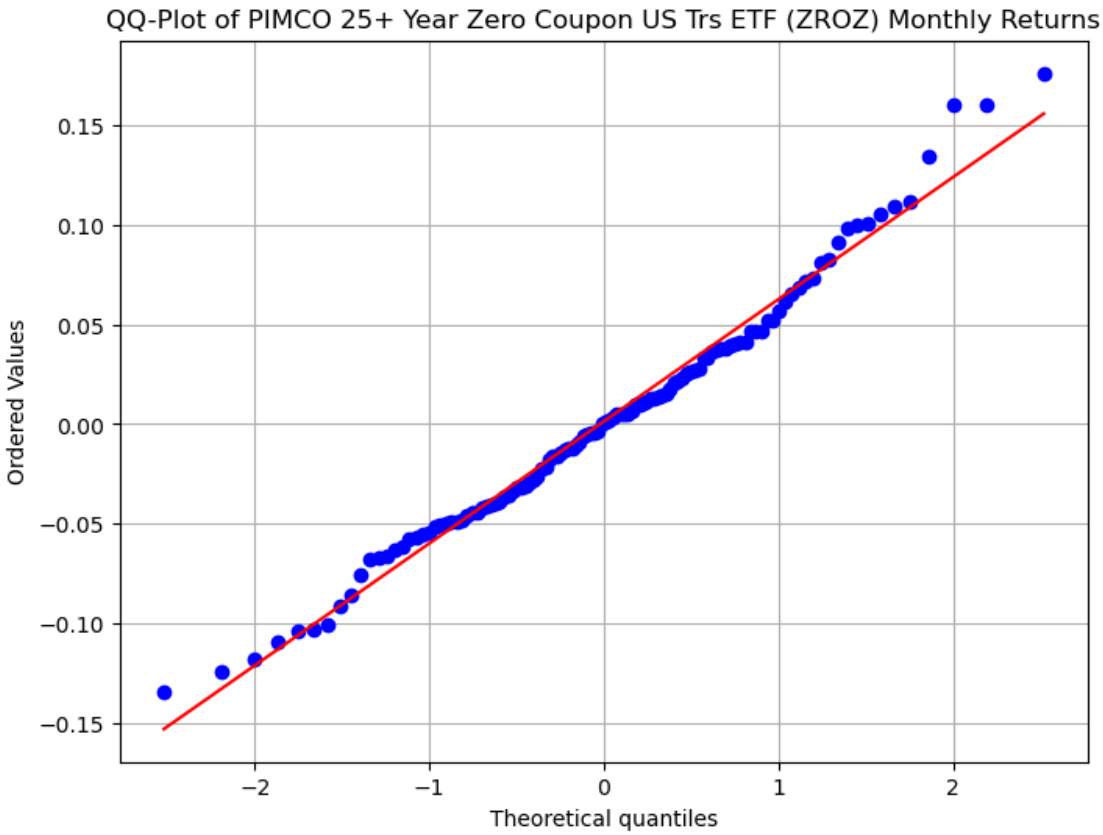
17 QQ Plots of Asset Returns (Figure 9)

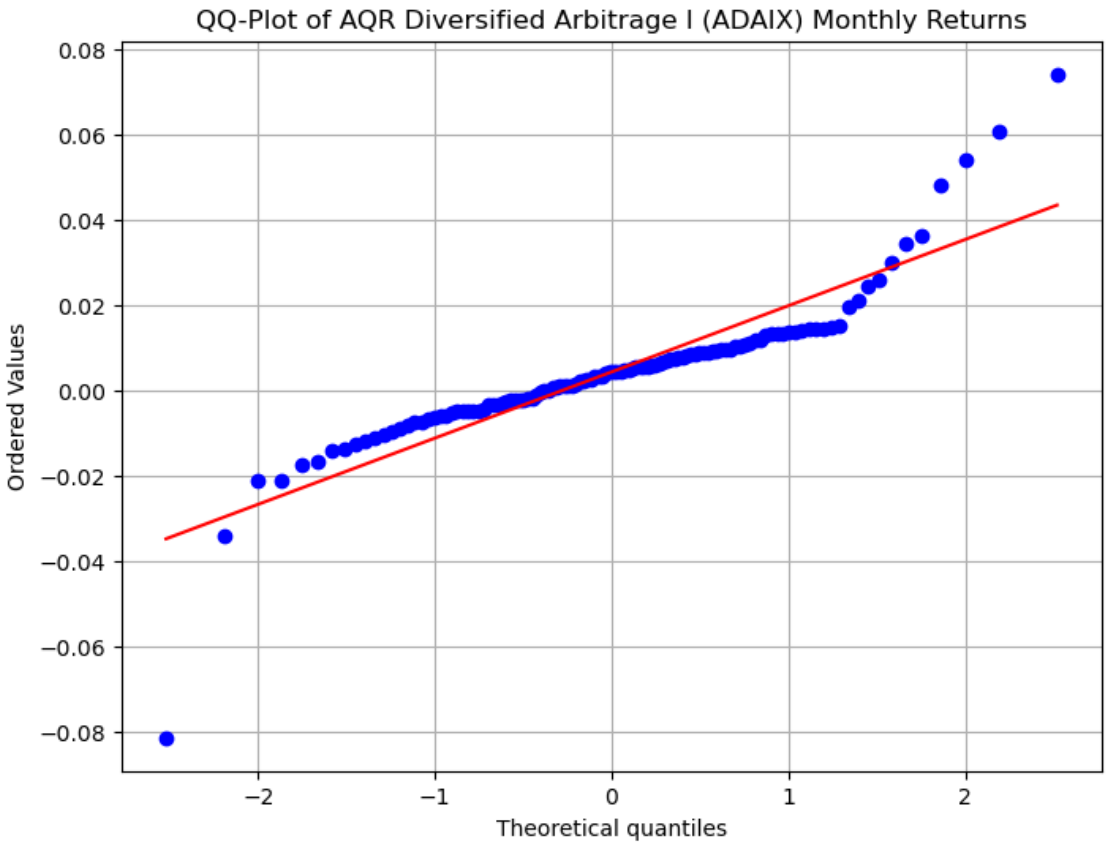
[24]:
```
import os
import matplotlib.pyplot as plt
import scipy.stats as stats

# 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()
```

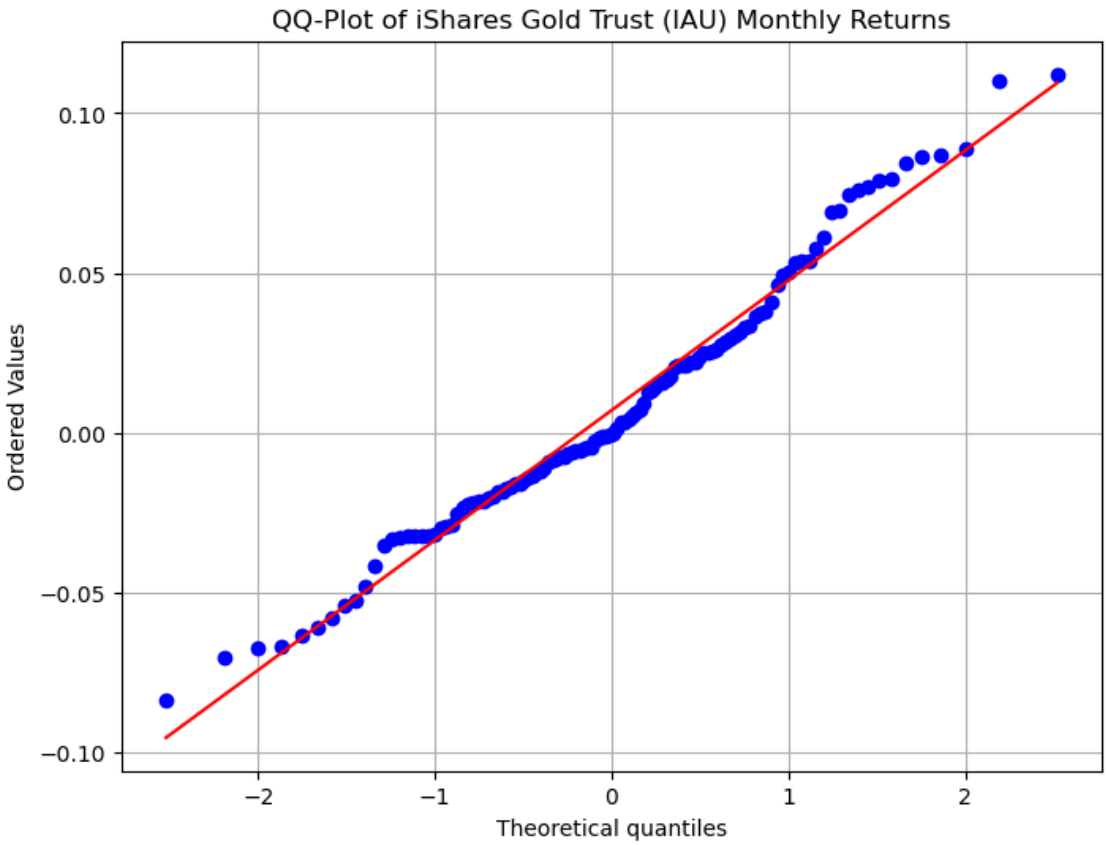
```
# Add each individual QQ plot to the combined figure without distortion
for i, (osm, osr, slope, intercept, r, asset) in enumerate(figures_data):
    ax = combined_axes[i]
    ax.plot(osm, osr, 'o')
    ax.plot(osm, slope * osm + intercept, 'r-', lw=2)
    ax.set_title(f"QQ-Plot of {asset} Monthly 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()
```

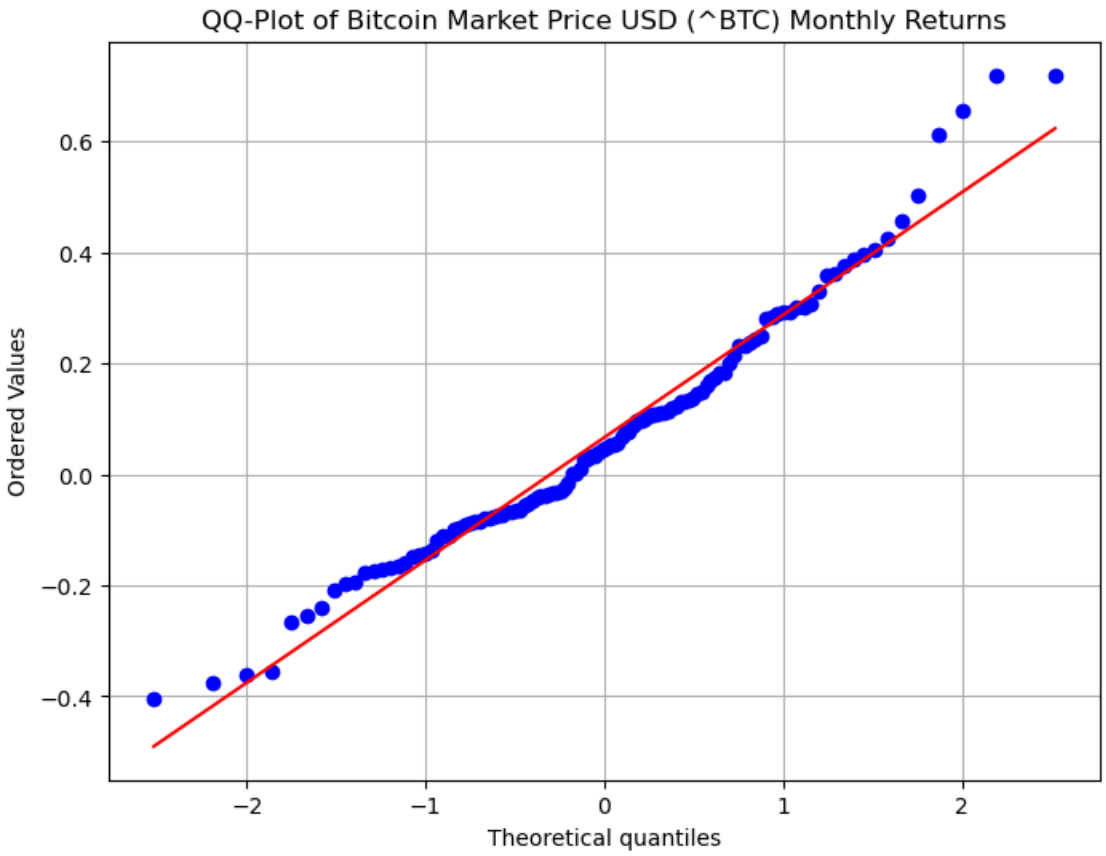

QQ-Plot of Vanguard LifeStrategy Income Fund (VASIX) Monthly Returns
Ordered Values
Theoretical quantiles
0.04
0.02
0.00
−0.02
−0.04
−2
−1
0
1
2

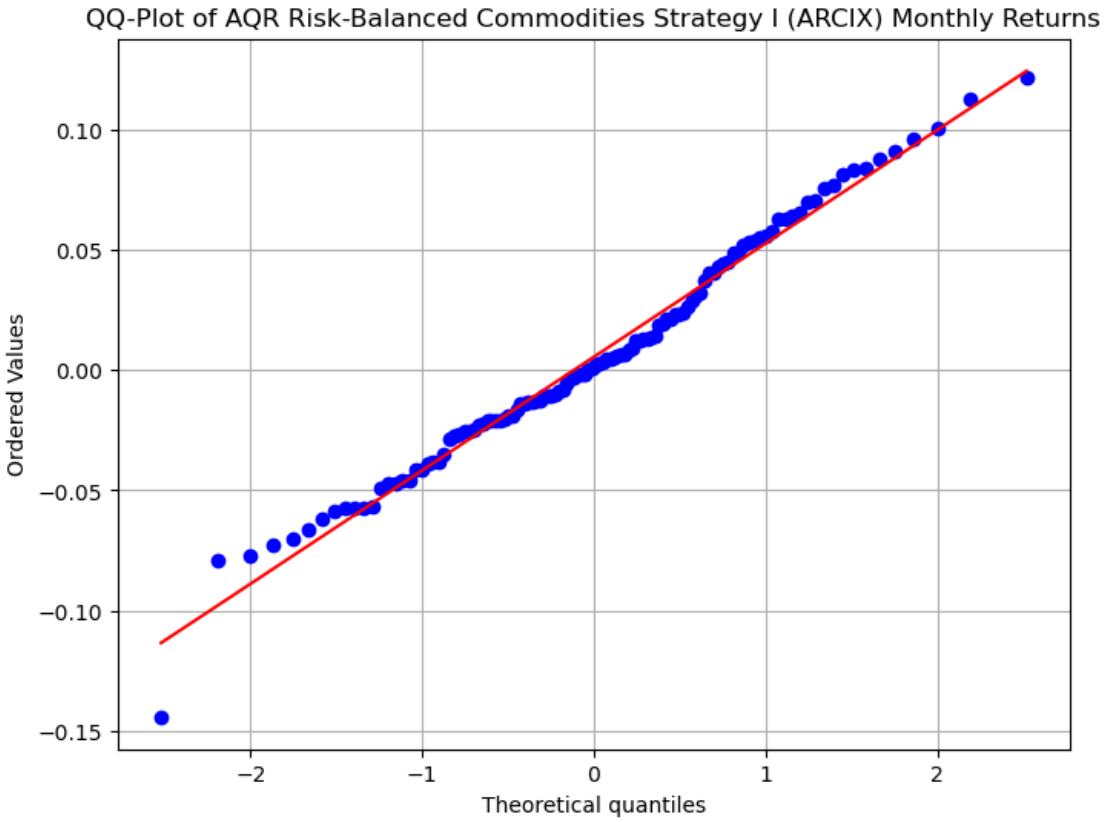

QQ-Plot of Vanguard Total World Stock ETF (VT) Monthly Returns
Ordered Values
Theoretical quantiles
0.10
0.05
0.00
−0.05
−0.10
−0.15
−2
−1
0
1
2

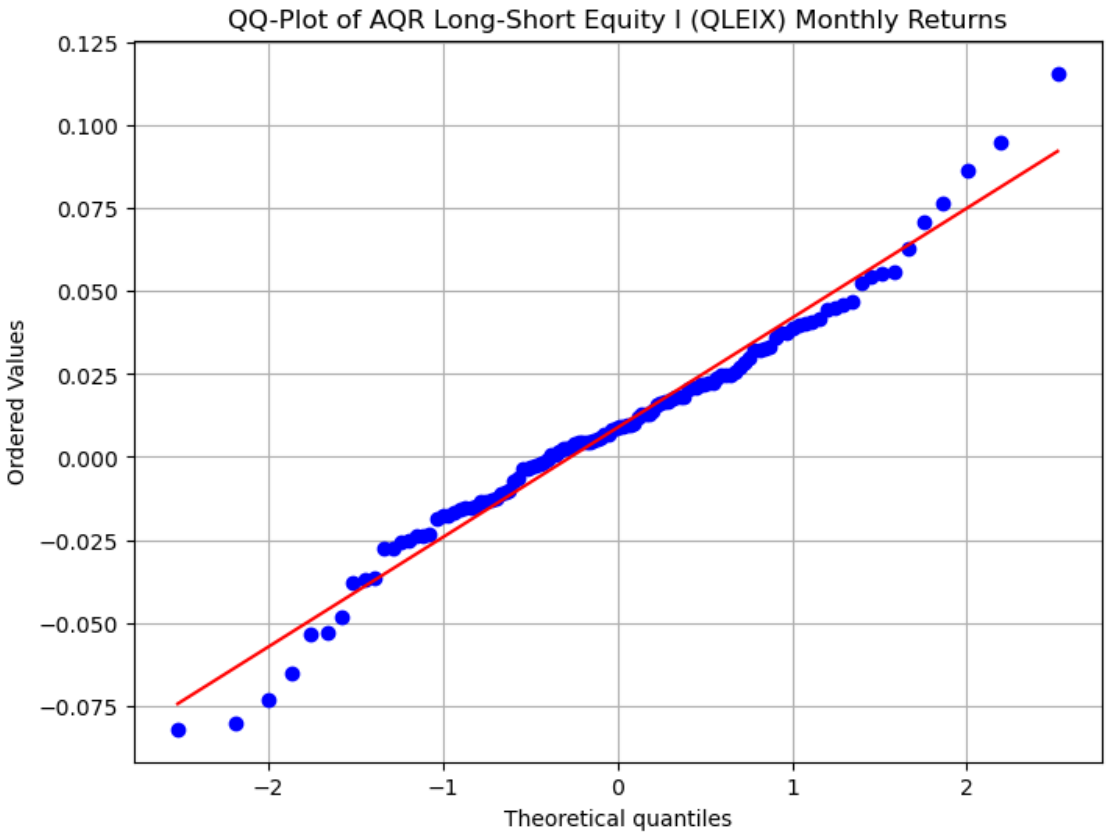

QQ-Plot of PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ) Monthly Returns
Ordered Values
Theoretical quantiles
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−2
−1
0
1
2

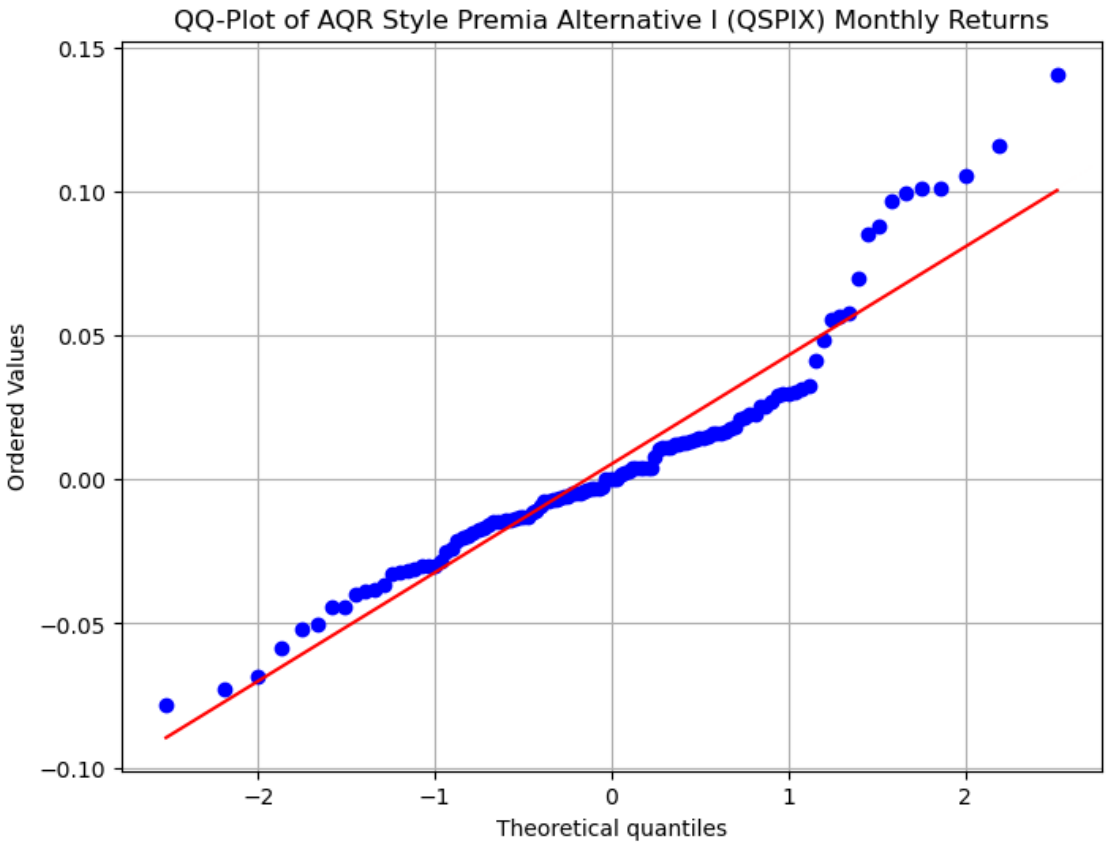

QQ-Plot of AQR Diversified Arbitrage I (ADAIX) Monthly Returns
Ordered Values
Theoretical quantiles
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
−0.08
−2
−1
0
1
2

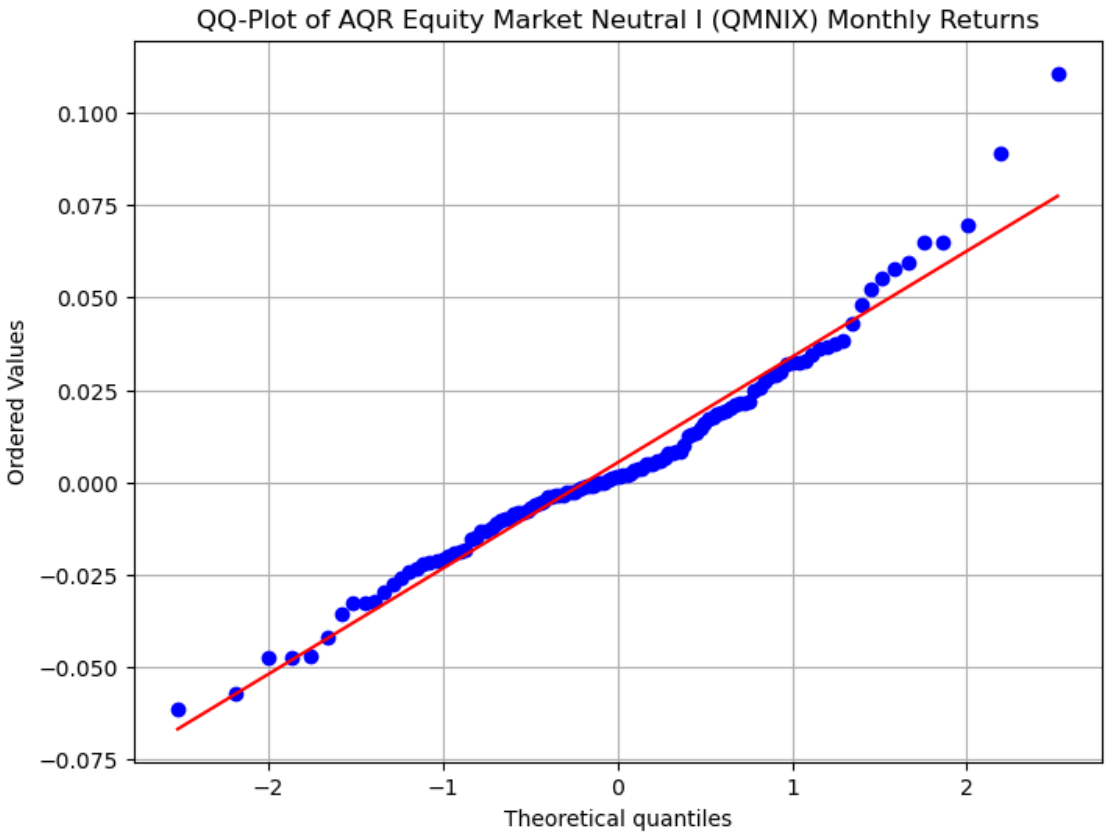

QQ-Plot of iShares Gold Trust (IAU) Monthly Returns
Ordered Values
Theoretical quantiles
0.10
0.05
0.00
−0.05
−0.10
−2
−1
0
1
2

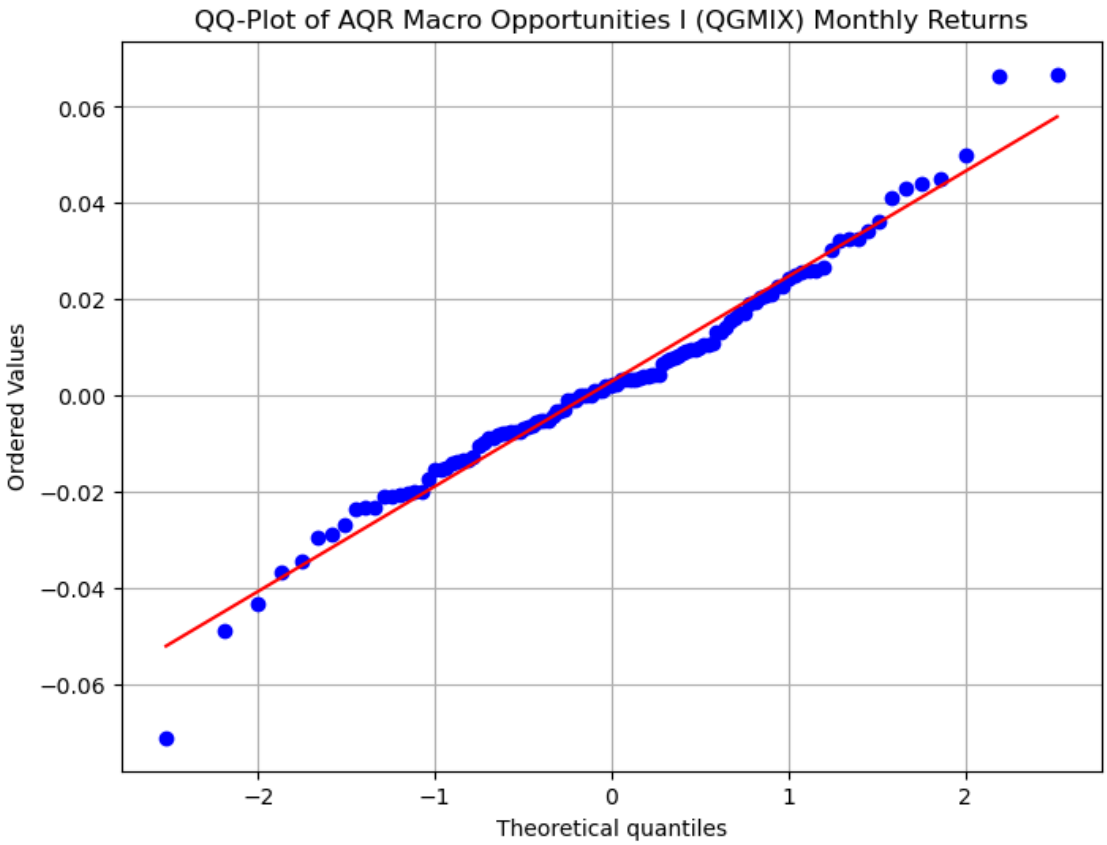

QQ-Plot of Bitcoin Market Price USD (^BTC) Monthly Returns
Ordered Values
Theoretical quantiles
0.6
0.4
0.2
0.0
−0.2
−0.4
−2
−1
0
1
2

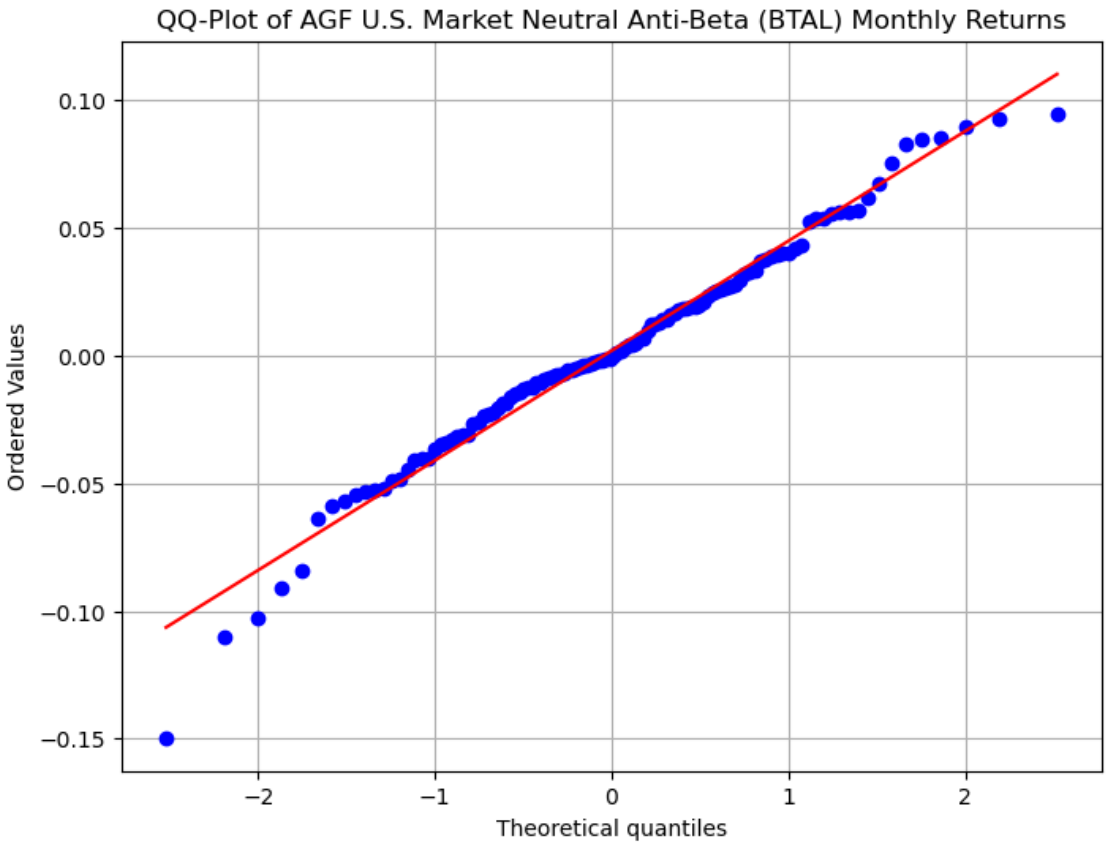

QQ-Plot of AQR Risk-Balanced Commodities Strategy I (ARCIX) Monthly Returns
Ordered Values
Theoretical quantiles
0.10
0.05
0.00
−0.05
−0.10
−0.15
−2
−1
0
1
2

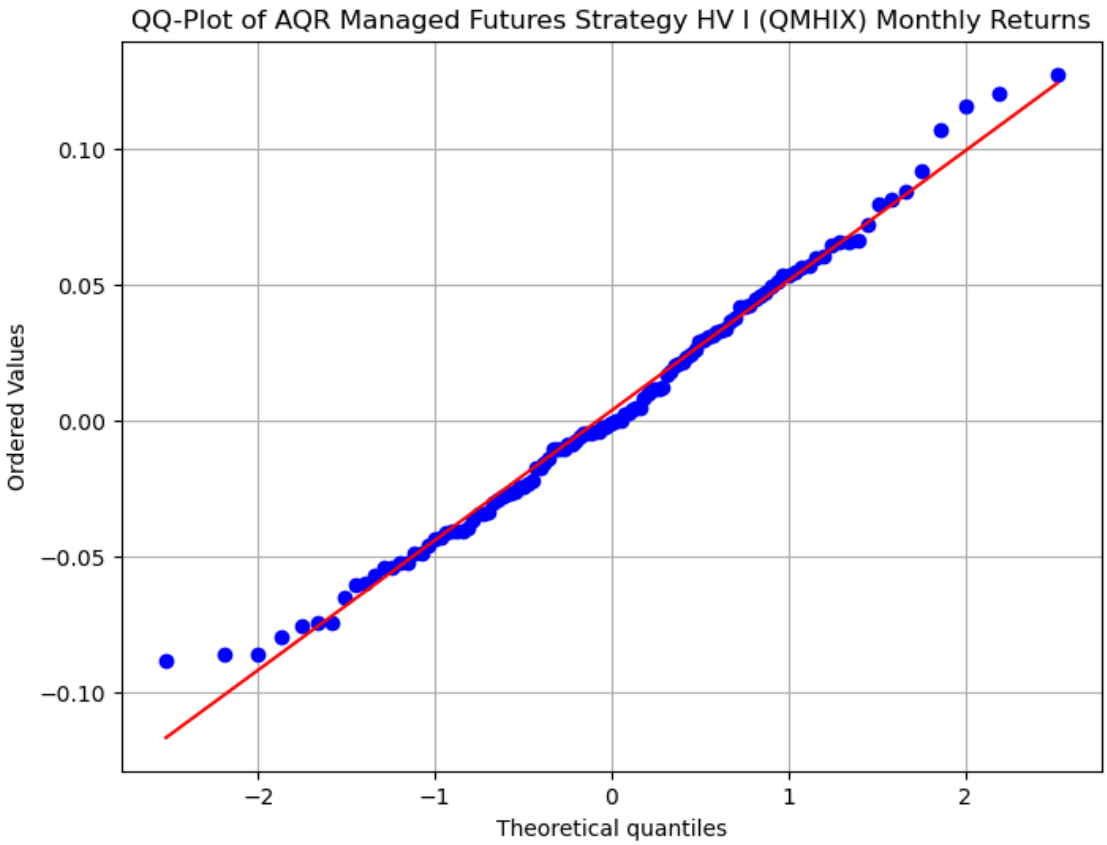

QQ-Plot of AQR Long-Short Equity I (QLEIX) Monthly Returns
Ordered Values
Theoretical quantiles
0.125
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−2
−1
0
1
2

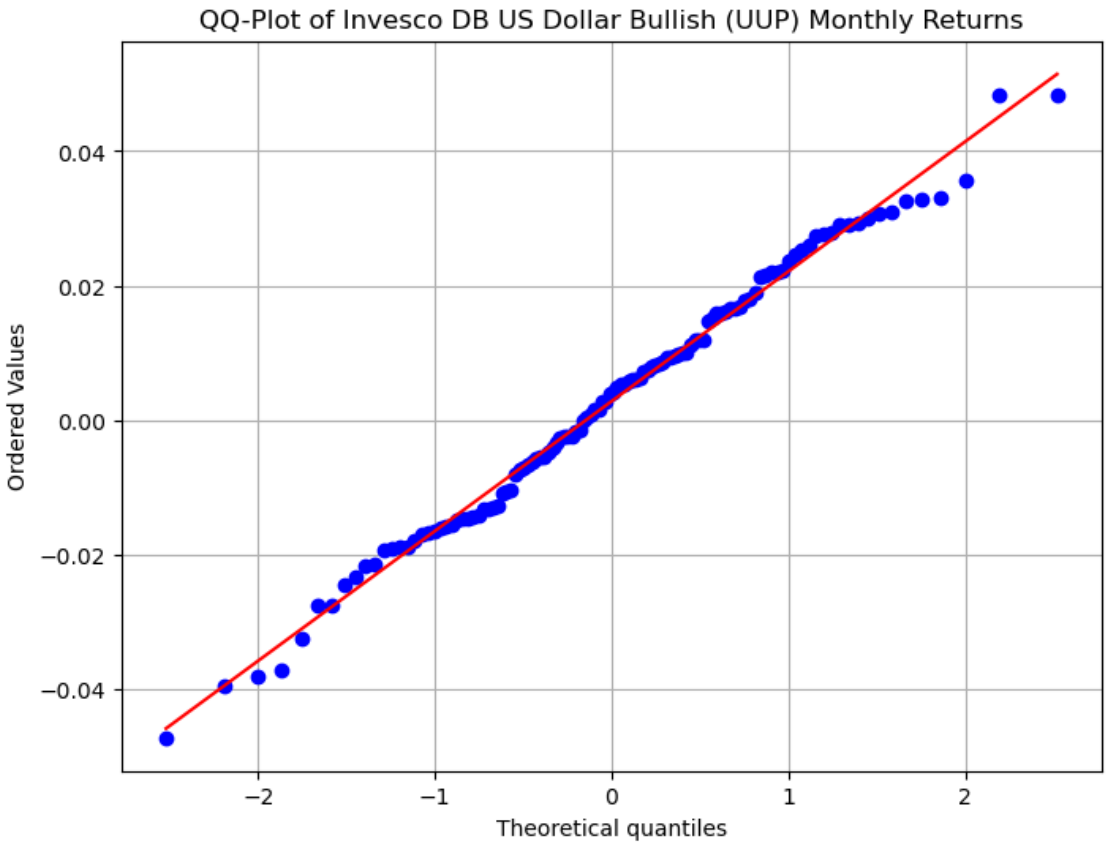

QQ-Plot of AQR Style Premia Alternative I (QSPIX) Monthly Returns
Ordered Values
Theoretical quantiles
0.15
0.10
0.05
0.00
−0.05
−0.10
−2
−1
0
1
2

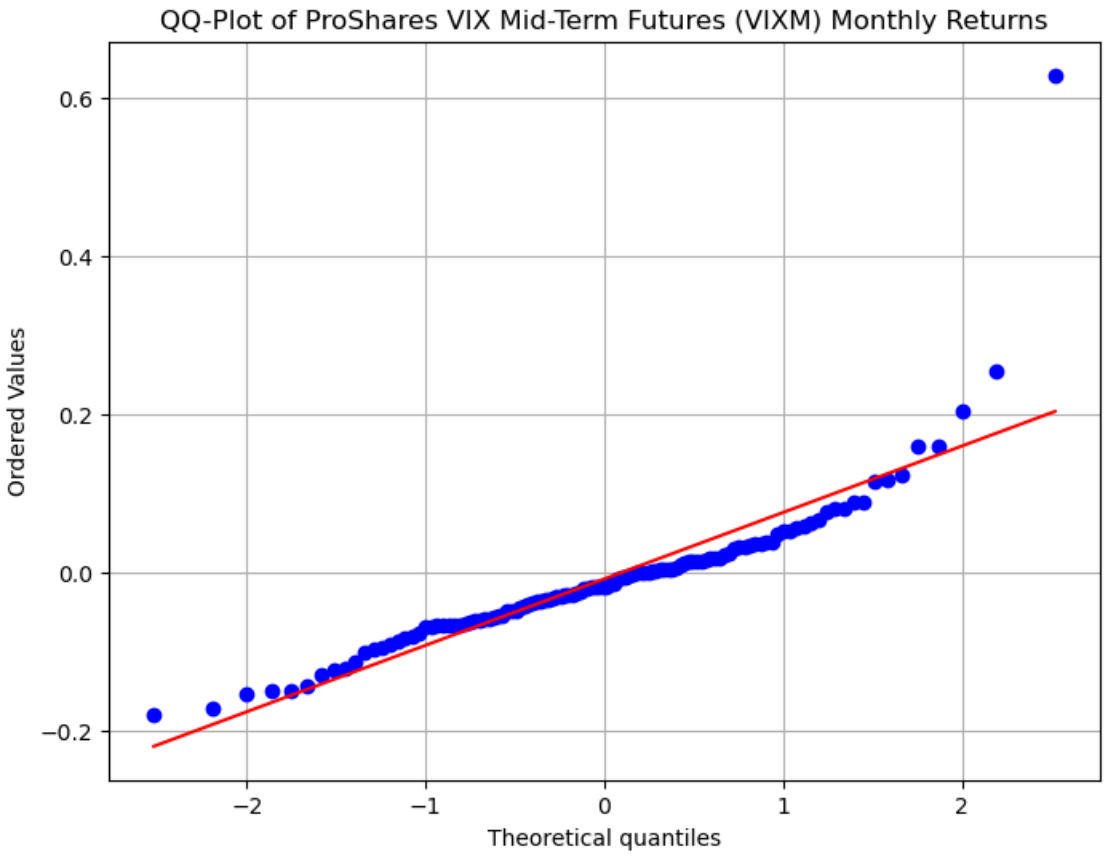

QQ-Plot of AQR Equity Market Neutral I (QMNIX) Monthly Returns
Ordered Values
Theoretical quantiles
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−2
−1
0
1
2

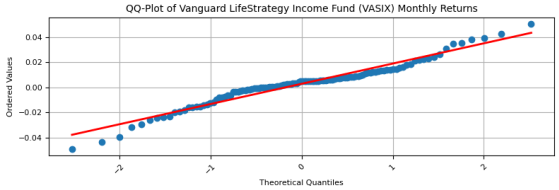

QQ-Plot of AQR Macro Opportunities I (QGMIX) Monthly Returns
Ordered Values
Theoretical quantiles
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
−2
−1
0
1
2

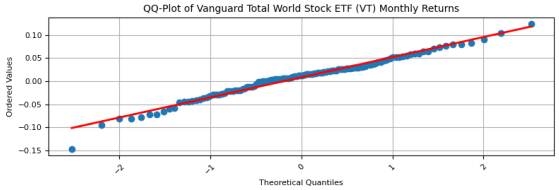

QQ-Plot of AGF U.S. Market Neutral Anti-Beta (BTAL) Monthly Returns
Ordered Values
Theoretical quantiles
0.10
0.05
0.00
−0.05
−0.10
−0.15
−2
−1
0
1
2

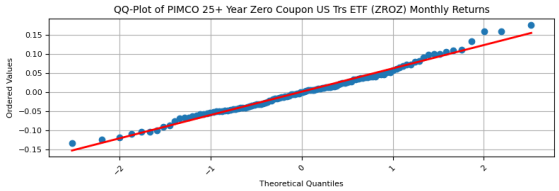

QQ-Plot of AQR Managed Futures Strategy HV I (QMHIX) Monthly Returns
Ordered Values
Theoretical quantiles
0.10
0.05
0.00
−0.05
−0.10
−2
−1
0
1
2

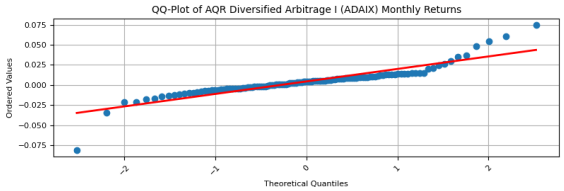

QQ-Plot of Invesco DB US Dollar Bullish (UUP) Monthly Returns
Ordered Values
Theoretical quantiles
0.04
0.02
0.00
−0.02
−0.04
−2
−1
0
1
2

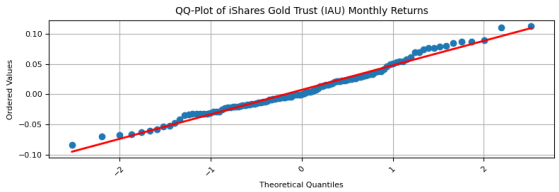

QQ-Plot of ProShares VIX Mid-Term Futures (VIXM) Monthly Returns
Ordered Values
Theoretical quantiles
0.6
0.4
0.2
0.0
−0.2
−2
−1
0
1
2

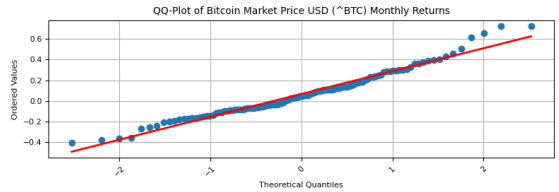

QQ-Plot of Vanguard LifeStrategy Income Fund (VASIX) Monthly Returns
Ordered Values
Theoretical Quantiles

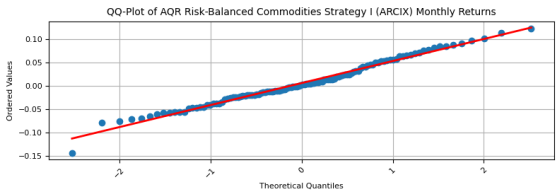

QQ-Plot of Vanguard Total World Stock ETF (VT) Monthly Returns
Ordered Values
Theoretical Quantiles

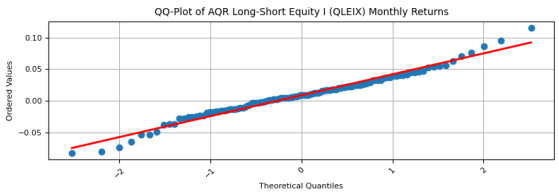

QQ-Plot of PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ) Monthly Returns
Ordered Values
Theoretical Quantiles

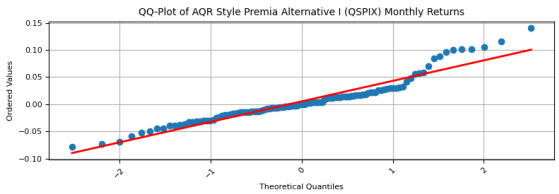

QQ-Plot of AQR Diversified Arbitrage I (ADAIX) Monthly Returns
Ordered Values
Theoretical Quantiles

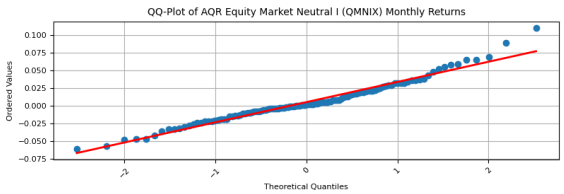

QQ-Plot of iShares Gold Trust (IAU) Monthly Returns
Ordered Values
Theoretical Quantiles

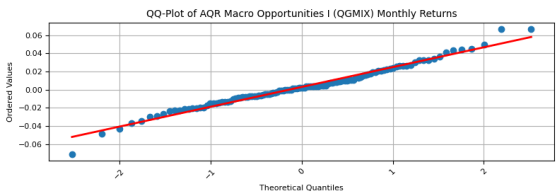

QQ-Plot of Bitcoin Market Price USD (^BTC) Monthly Returns
Ordered Values
Theoretical Quantiles

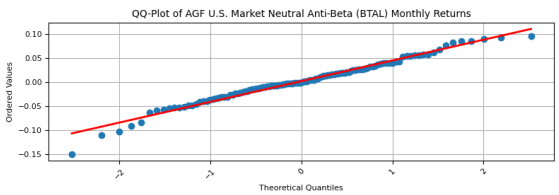

QQ-Plot of AQR Risk-Balanced Commodities Strategy I (ARCIX) Monthly Returns
Ordered Values
Theoretical Quantiles

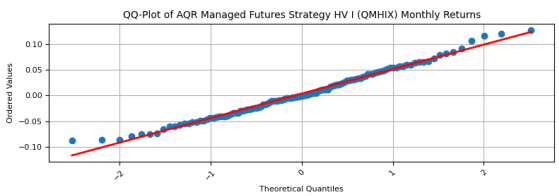

QQ-Plot of AQR Long-Short Equity I (QLEIX) Monthly Returns
Ordered Values
Theoretical Quantiles

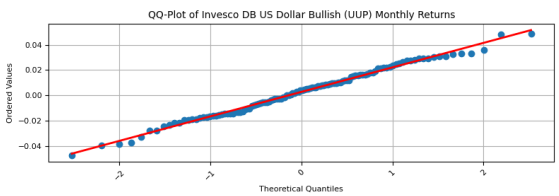

QQ-Plot of AQR Style Premia Alternative I (QSPIX) Monthly Returns
Ordered Values
Theoretical Quantiles

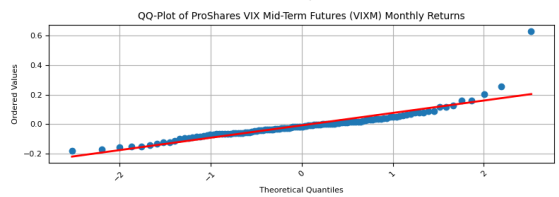

QQ-Plot of AQR Equity Market Neutral I (QMNIX) Monthly Returns
Ordered Values
Theoretical Quantiles


QQ-Plot of AQR Macro Opportunities I (QGMIX) Monthly Returns
Ordered Values
Theoretical Quantiles


QQ-Plot of AGF U.S. Market Neutral Anti-Beta (BTAL) Monthly Returns
Ordered Values
Theoretical Quantiles


QQ-Plot of AQR Managed Futures Strategy HV I (QMHIX) Monthly Returns
Ordered Values
Theoretical Quantiles


QQ-Plot of Invesco DB US Dollar Bullish (UUP) Monthly Returns
Ordered Values
Theoretical Quantiles


QQ-Plot of ProShares VIX Mid-Term Futures (VIXM) Monthly Returns
Ordered Values
Theoretical Quantiles

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)

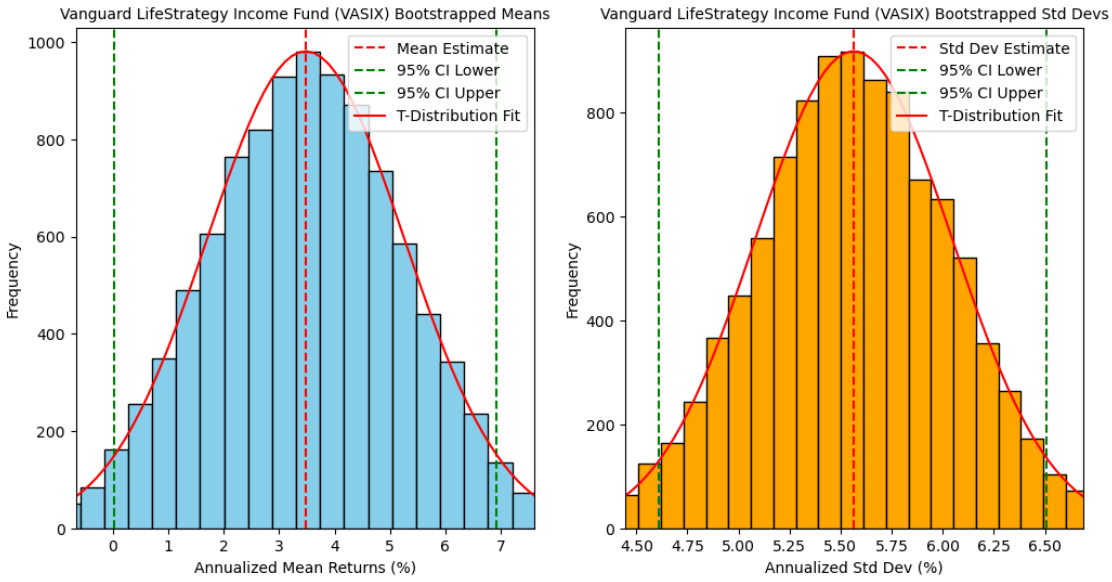
[30]:
```
import os
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import t
from PIL import Image

# 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)
```

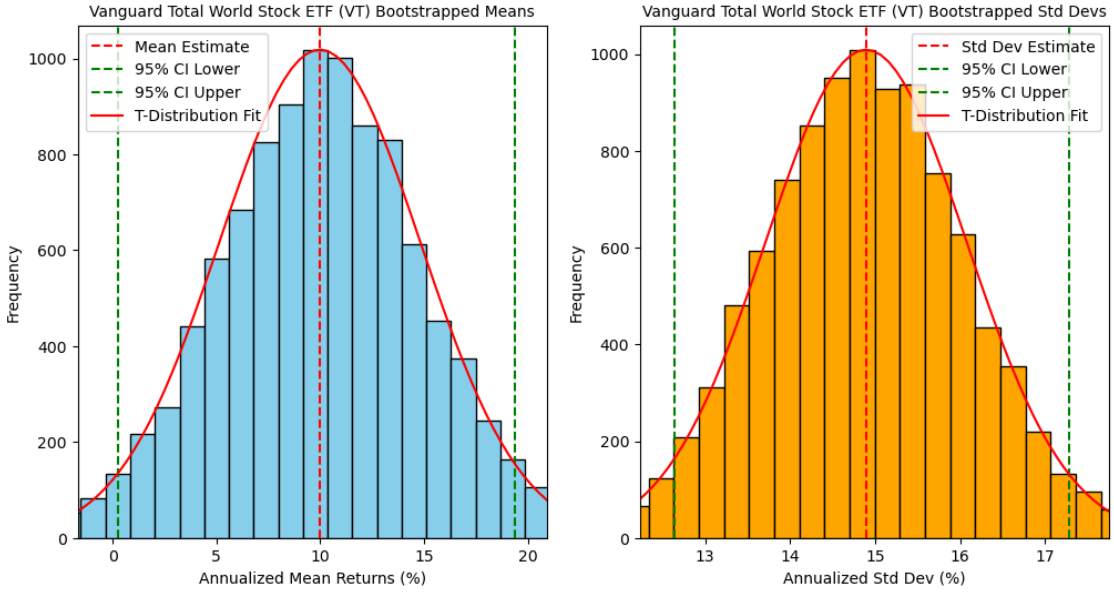
```
    ax_means = axes[0]
    count, bins, _ = ax_means.hist(annualized_bootstrap_means, bins=30,␣
↪color='skyblue', edgecolor='black')  # Get histogram counts
    ax_means.axvline(np.mean(annualized_bootstrap_means), color='red',␣
↪linestyle='--', label="Mean Estimate")
    ax_means.axvline(mean_ci_lower,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

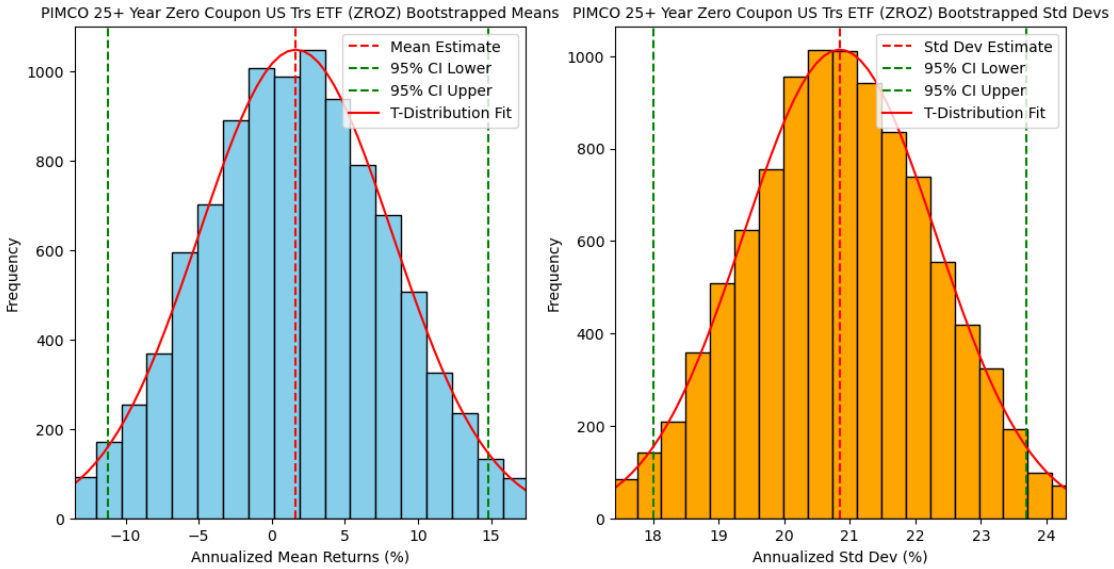
Bootstrap Analysis for Vanguard LifeStrategy Income Fund (VASIX)



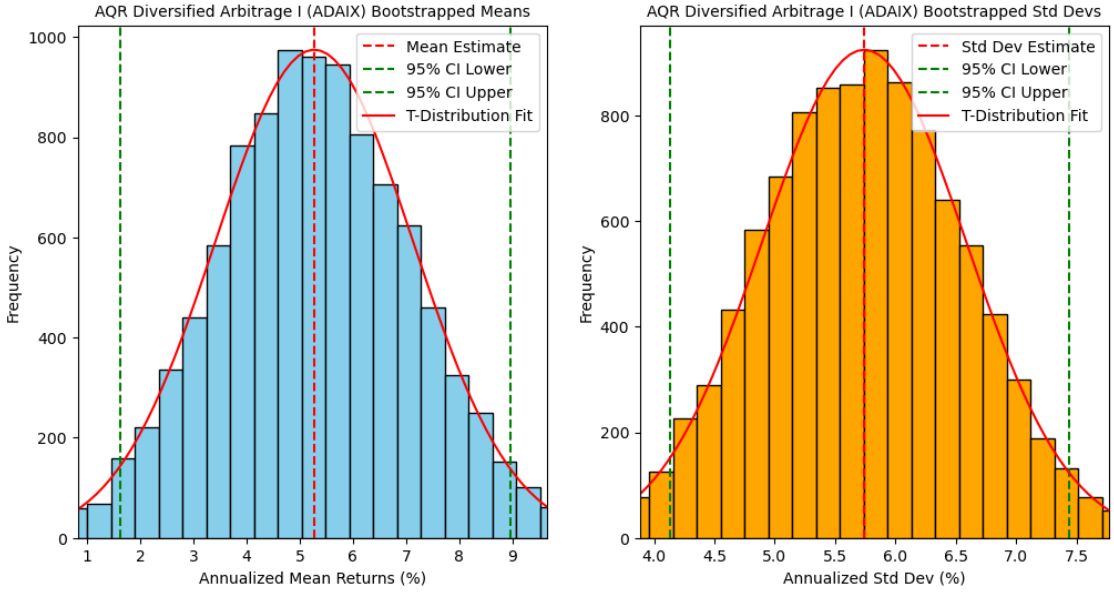
Bootstrap Analysis for Vanguard Total World Stock ETF (VT)



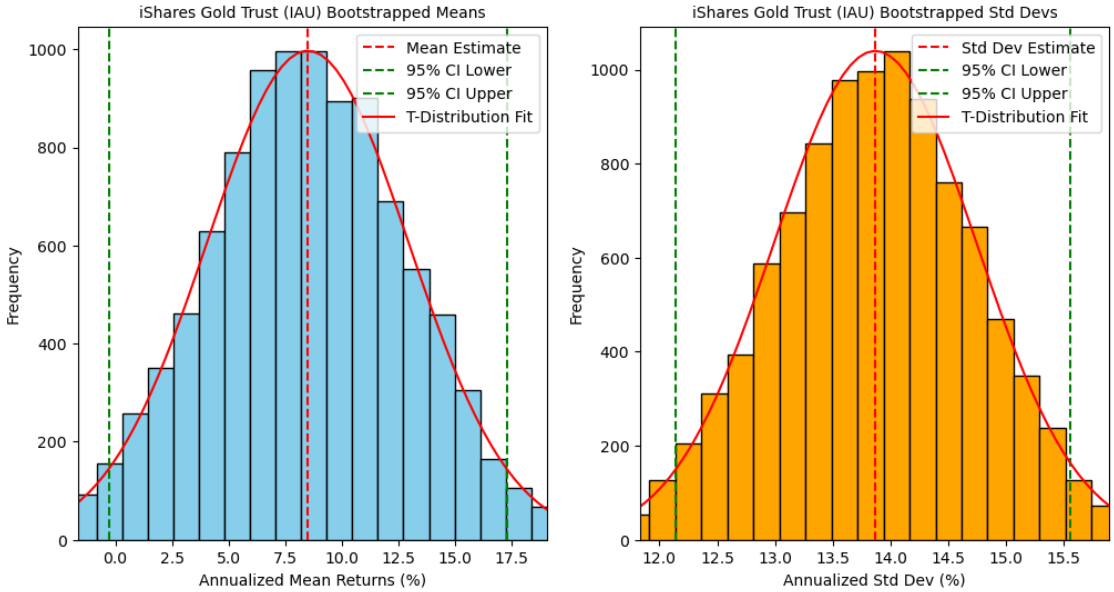
Bootstrap Analysis for PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ)



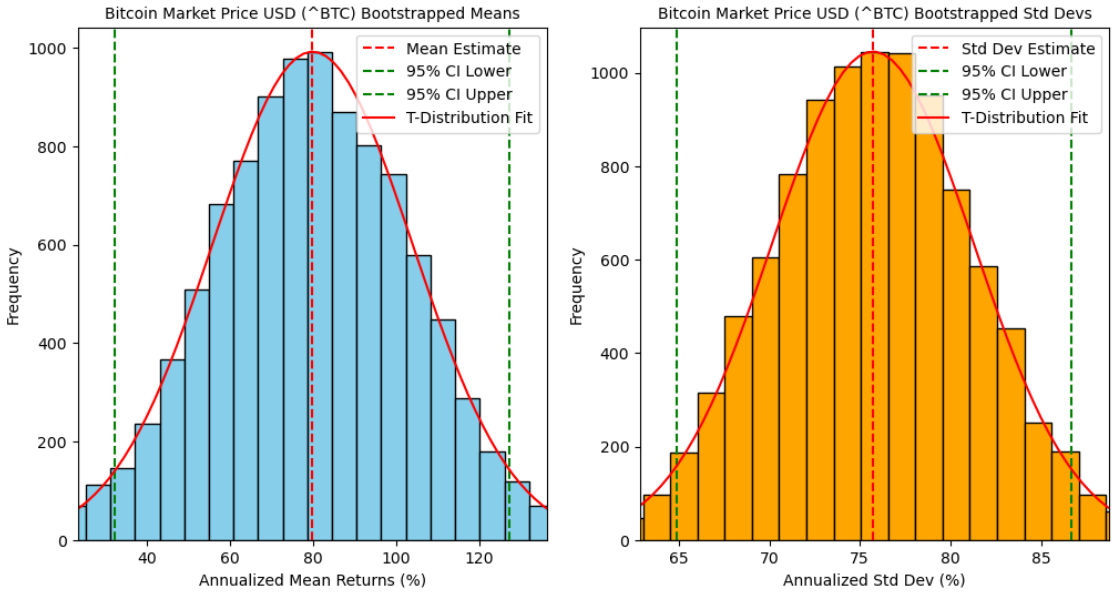
Bootstrap Analysis for AQR Diversified Arbitrage I (ADAIX)

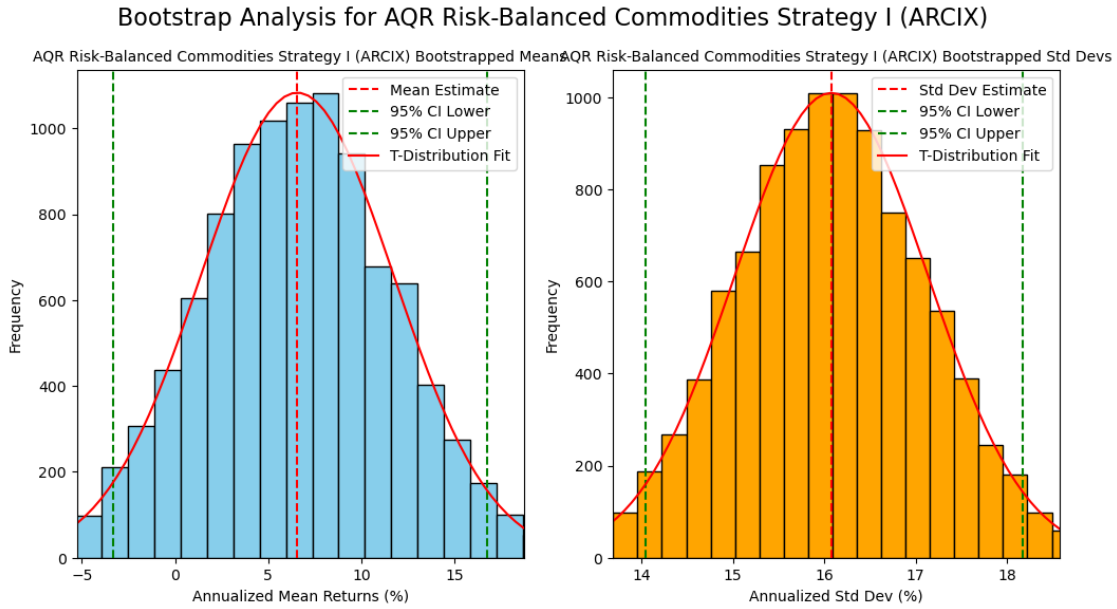


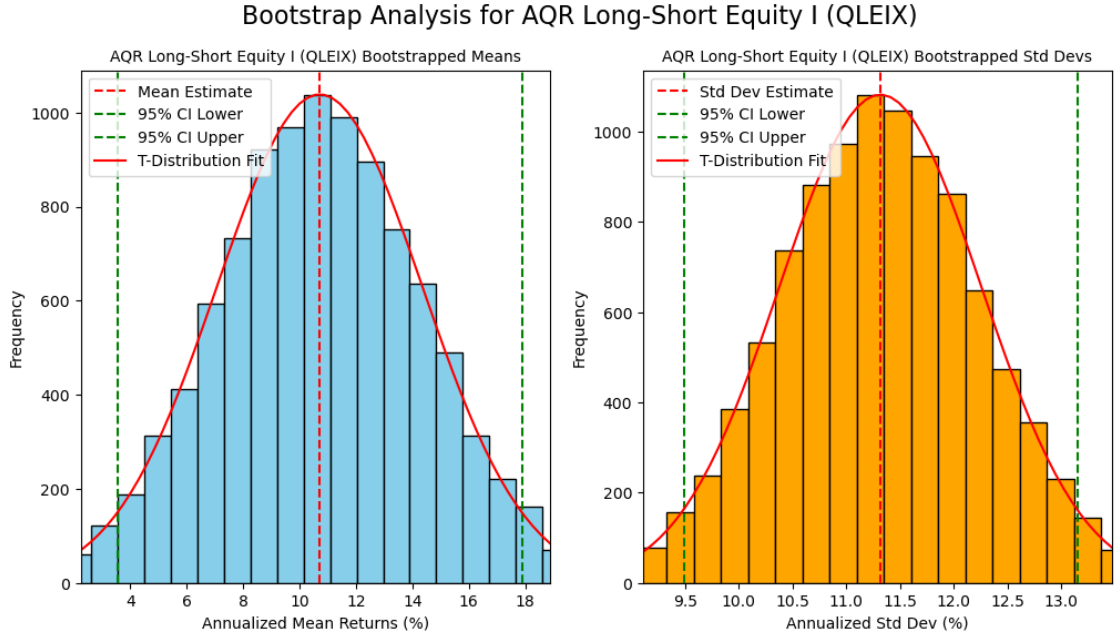
Bootstrap Analysis for iShares Gold Trust (IAU)



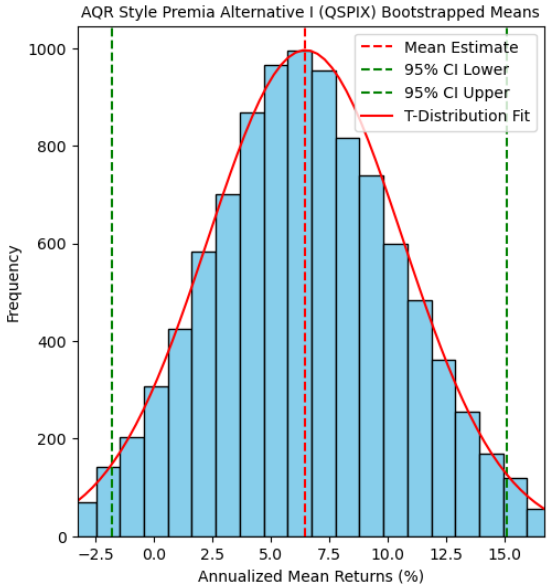
Bootstrap Analysis for Bitcoin Market Price USD (^BTC)

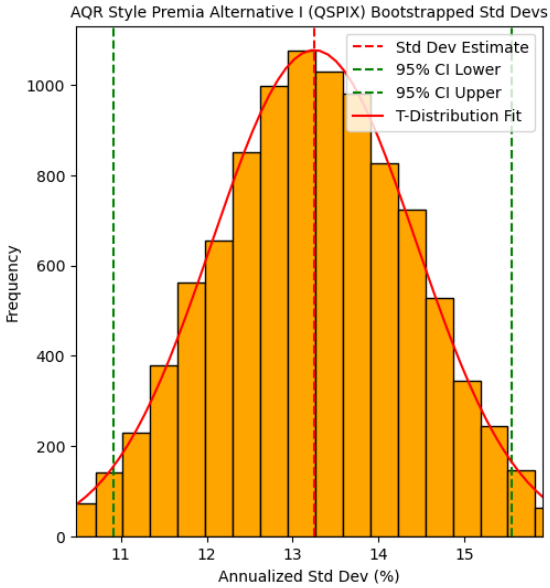



Bootstrap Analysis for AQR Risk-Balanced Commodities Strategy I (ARCIX)
AQR Risk-Balanced Commodities Strategy I (ARCIX) Bootstrapped Means
Mean Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Mean Returns (%)
AQR Risk-Balanced Commodities Strategy I (ARCIX) Bootstrapped Std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

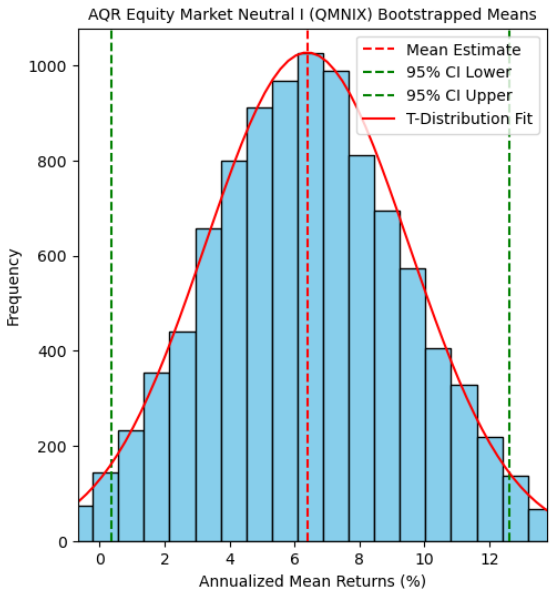

Bootstrap Analysis for AQR Long-Short Equity I (QLEIX)
AQR Long-Short Equity I (QLEIX) Bootstrapped Means
Mean Estimate
95% CI Lower
95% CI Upper
T-Distribution Fit
Frequency
Annualized Mean Returns (%)
AQR Long-Short Equity I (QLEIX) Bootstrapped Std Devs
Std Dev Estimate
95% CI Lower
95% CI Upper
T-Distribution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

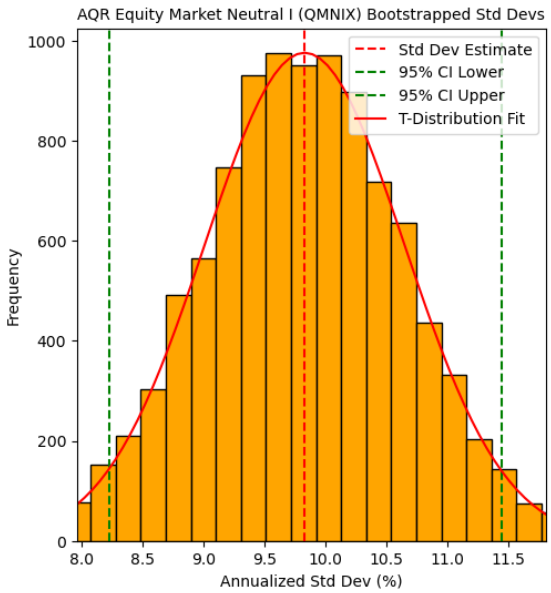
Bootstrap Analysis for AQR Style Premia Alternative I (QSPIX)



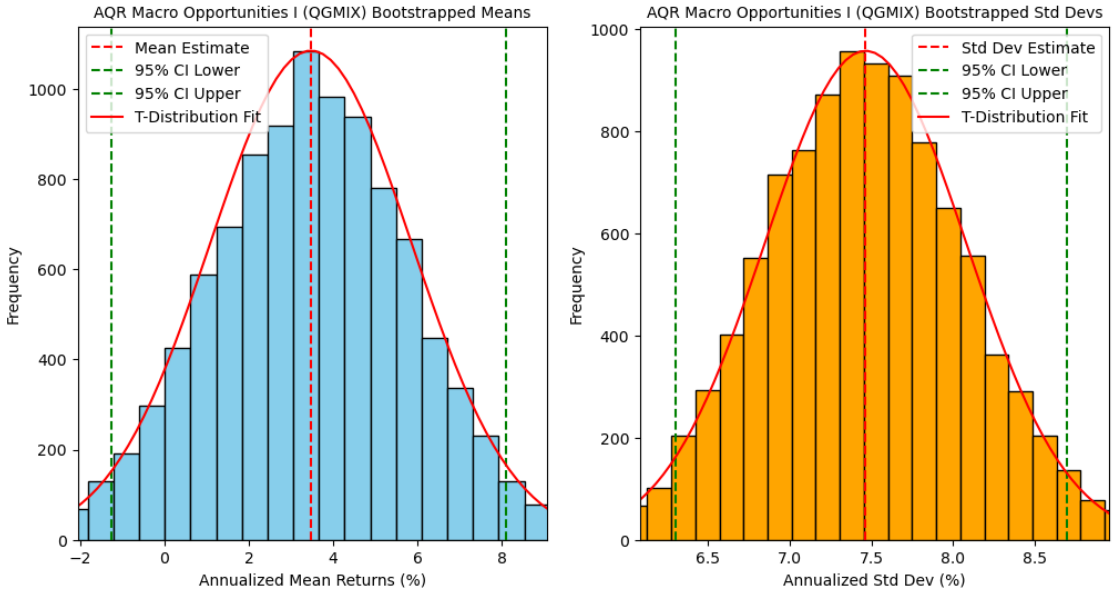


Bootstrap Analysis for AQR Equity Market Neutral I (QMNIX)

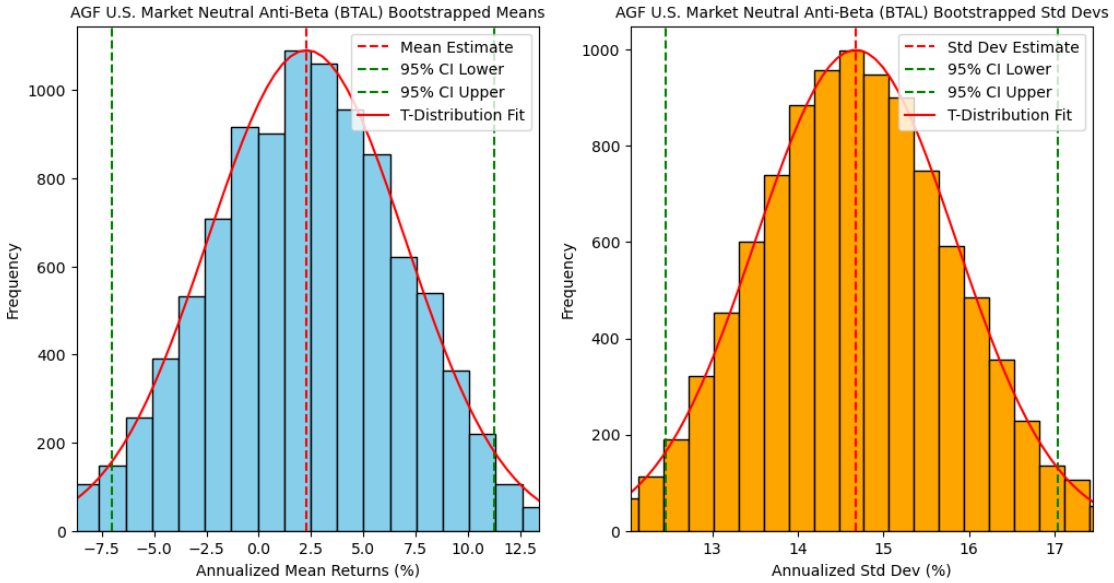




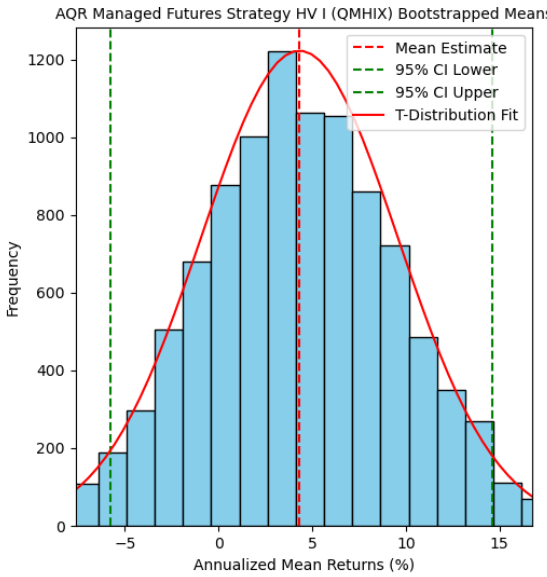
Bootstrap Analysis for AQR Macro Opportunities I (QGMIX)

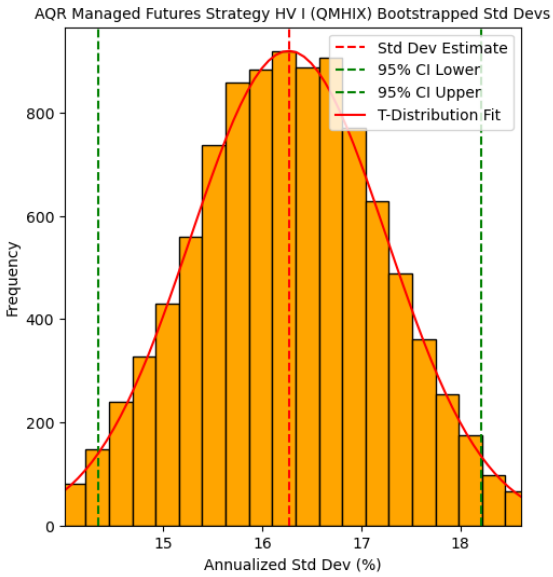


Bootstrap Analysis for AGF U.S. Market Neutral Anti-Beta (BTAL)

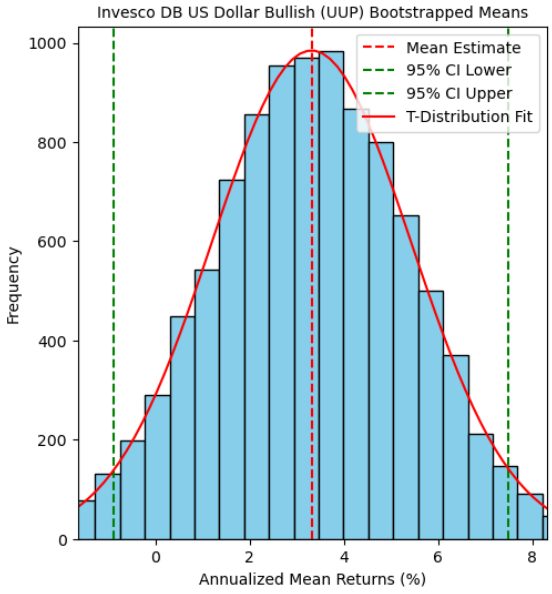


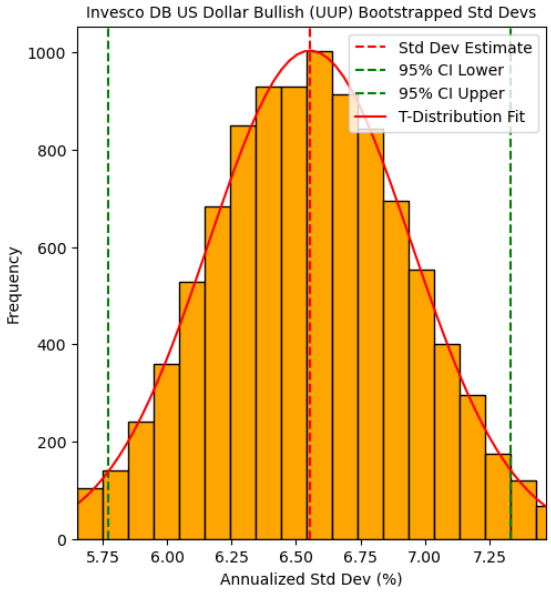
Bootstrap Analysis for AQR Managed Futures Strategy HV I (QMHIX)

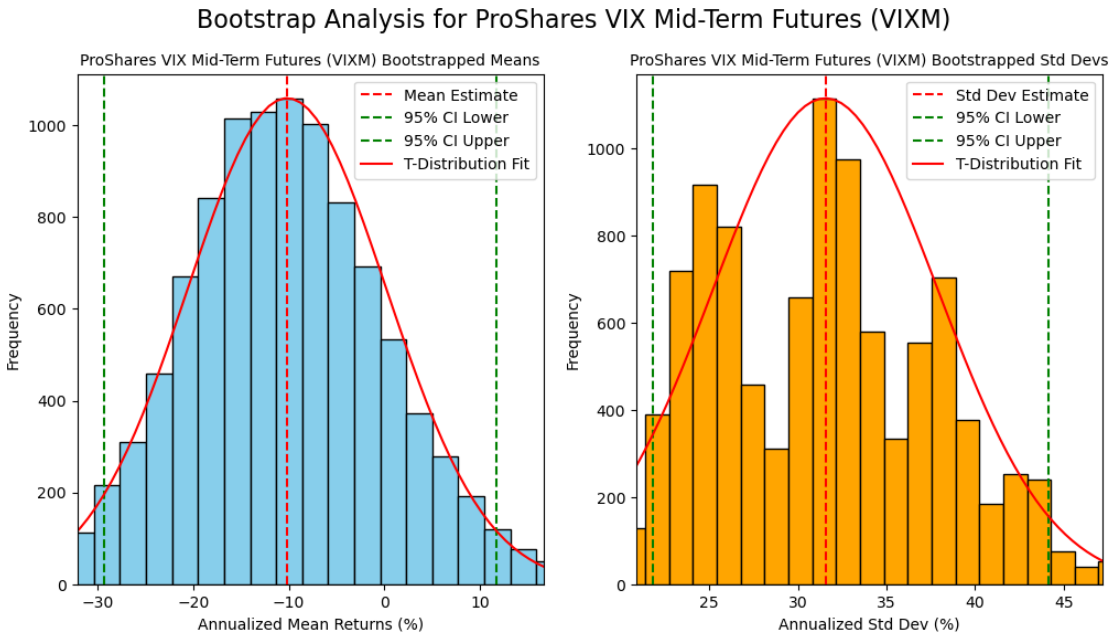




Bootstrap Analysis for Invesco DB US Dollar Bullish (UUP)







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')
```

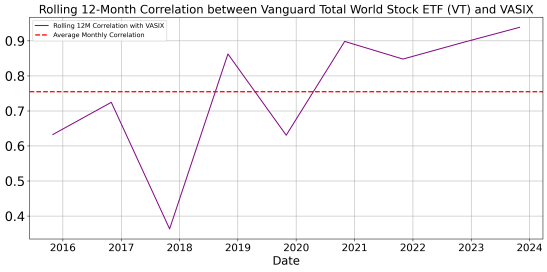
```python
# Sort the index to ensure chronological order
data = data.sort_index()

# ------------------------------ #
#        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
    """
```

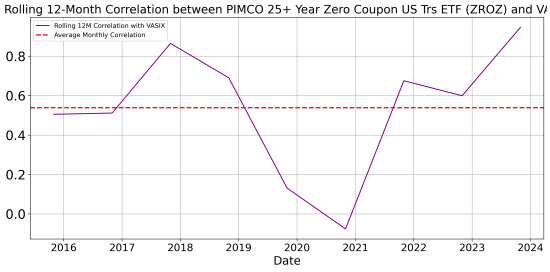
```
    # Drop NaN values resulting from rolling window
    rolling_metric = rolling_metric.dropna()

    # 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()

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

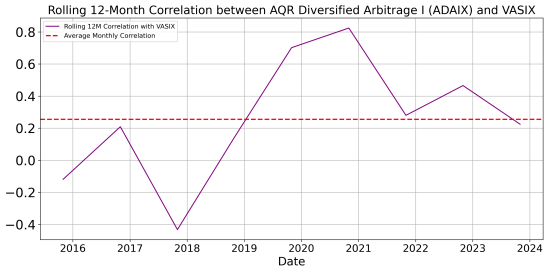
```
#    Define Plotting Function    #
# ------------------------------ #

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:
```

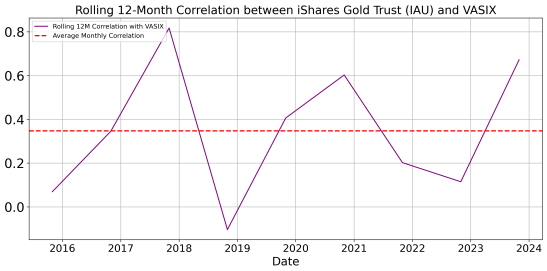
```
    # Extract the downsampled rolling correlation series for the asset
    corr_series = downsampled_correlations[asset].dropna()

    # 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))
```

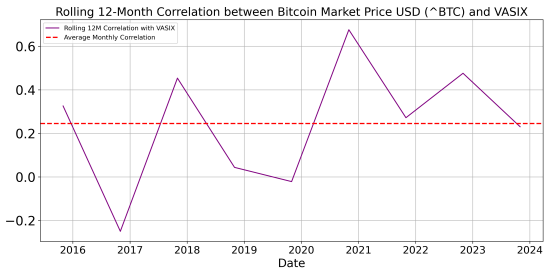
```
# Save and display the combined image
combined_image_filename = f"Combined_Rolling12M_Correlation_vs_{benchmark_label}.png"
combined_metrics_image_path = os.path.join(output_folder, combined_image_filename)
combined_metrics_image.save(combined_metrics_image_path, format='PNG')
display(combined_metrics_image)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

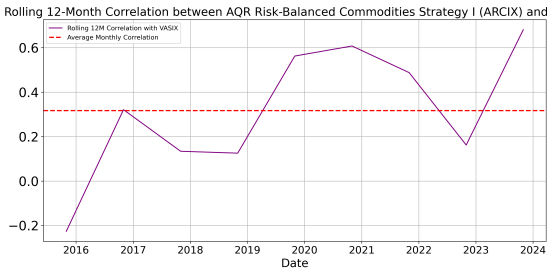

Rolling 12-Month Correlation between Vanguard Total World Stock ETF (VT) and VASIX
Rolling 12M Correlation with VASIX
Average Monthly Correlation
Date

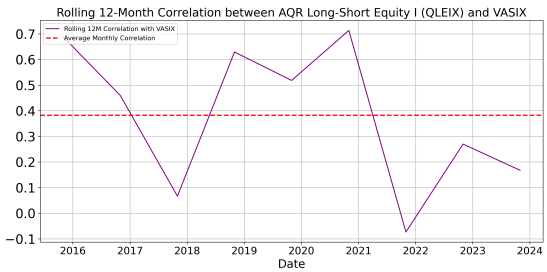

Rolling 12-Month Correlation between PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ) and VA
Rolling 12M Correlation with VASIX
Average Monthly Correlation
Date

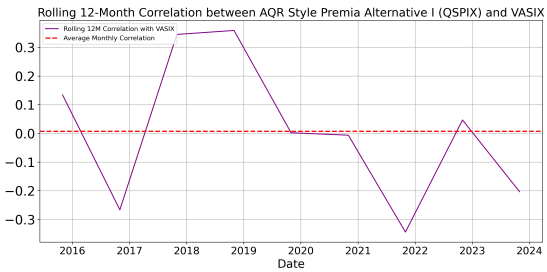

Rolling 12-Month Correlation between AQR Diversified Arbitrage I (ADAIX) and VASIX
Rolling 12M Correlation with VASIX
Average Monthly Correlation
Date

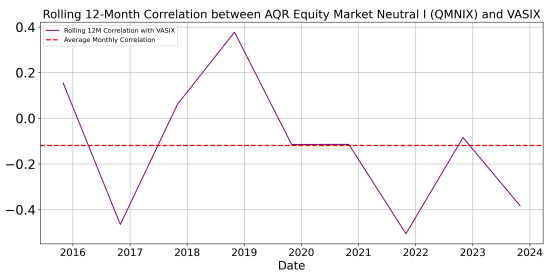

Rolling 12-Month Correlation between iShares Gold Trust (IAU) and VASIX
Rolling 12M Correlation with VASIX
Average Monthly Correlation
Date

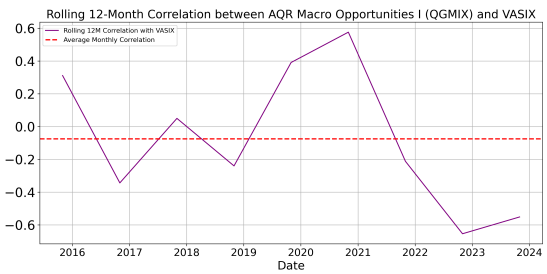

Rolling 12-Month Correlation between Bitcoin Market Price USD (^BTC) and VASIX
Rolling 12M Correlation with VASIX
Average Monthly Correlation
Date

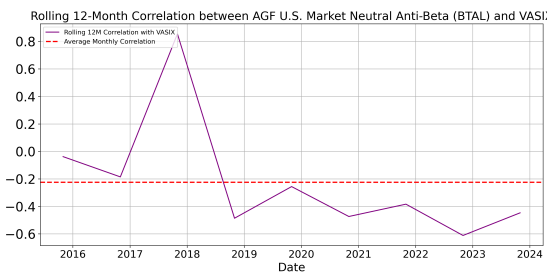

Rolling 12-Month Correlation between AQR Risk-Balanced Commodities Strategy I (ARCIX) and
Rolling 12M Correlation with VASIX
Average Monthly Correlation
Date

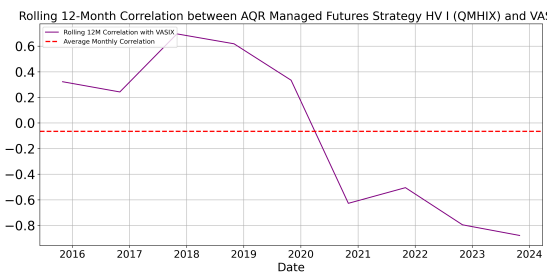

Rolling 12-Month Correlation between AQR Long-Short Equity I (QLEIX) and VASIX
Rolling 12M Correlation with VASIX
Average Monthly Correlation
Date

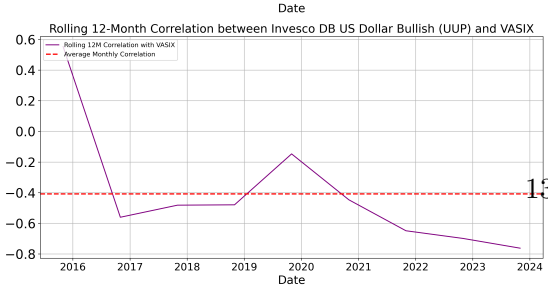

Rolling 12-Month Correlation between AQR Style Premia Alternative I (QSPIX) and VASIX
Rolling 12M Correlation with VASIX
Average Monthly Correlation
Date

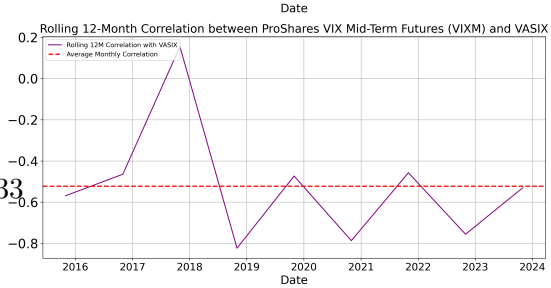

Rolling 12-Month Correlation between AQR Equity Market Neutral I (QMNIX) and VASIX
Rolling 12M Correlation with VASIX
Average Monthly Correlation
Date


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)

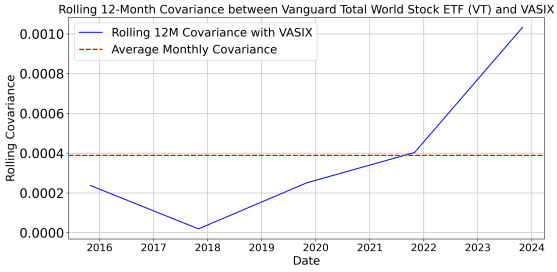
[32]:
```
import os
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from PIL import Image
from IPython.display import display
from scipy import stats

# ------------------------------ #
#         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()
```

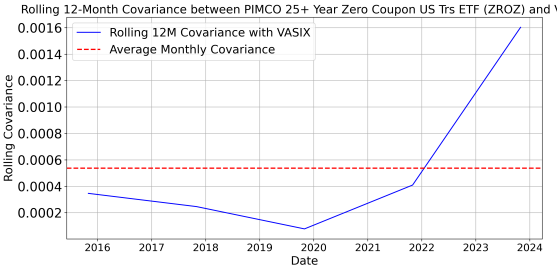
```
# ------------------------------ #
#      Prepare Output Folder     #
# ------------------------------ #

# 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]
```

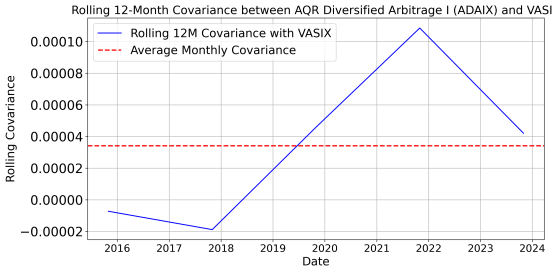
```
# Initialize a dictionary to store rolling covariances
rolling_covariances = pd.DataFrame()

# 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))
```

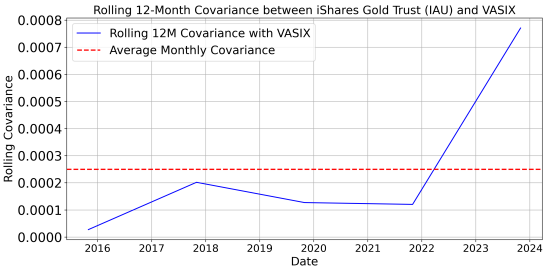
```
    plt.plot(cov_series.index, cov_series, color='blue', label=f'Rolling 12M␣
↪Covariance with {benchmark_label}')

    # 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
```

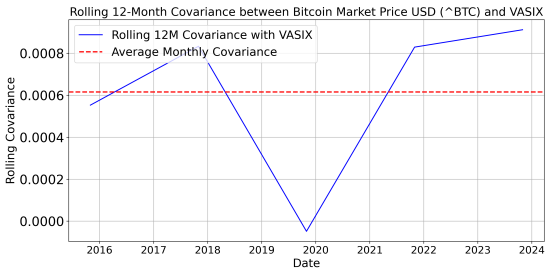
```
plot_filenames = [f"{asset}_Rolling12M_Covariance_with_{benchmark_label}.png"␣
 ↪for asset in assets]

# 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)
```

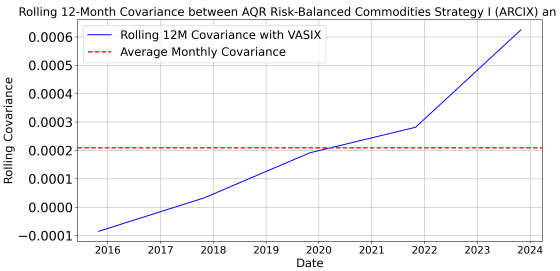

Rolling 12-Month Covariance between Vanguard Total World Stock ETF (VT) and VASIX
Rolling 12M Covariance with VASIX
Average Monthly Covariance
Rolling Covariance
Date

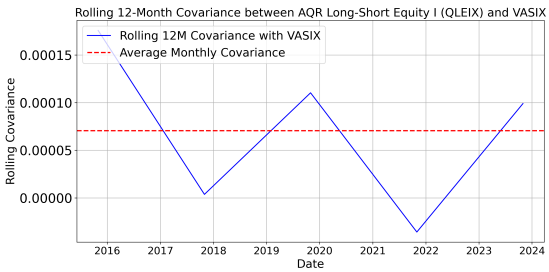

Rolling 12-Month Covariance between PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ) and
Rolling 12M Covariance with VASIX
Average Monthly Covariance
Rolling Covariance
Date

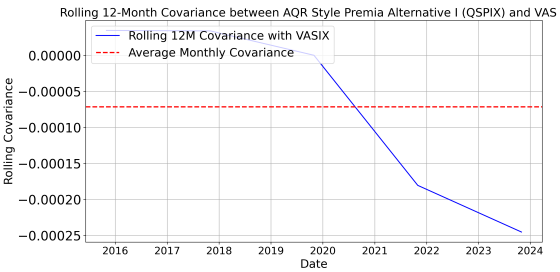

Rolling 12-Month Covariance between AQR Diversified Arbitrage I (ADAIX) and VASIX
Rolling 12M Covariance with VASIX
Average Monthly Covariance
Rolling Covariance
Date

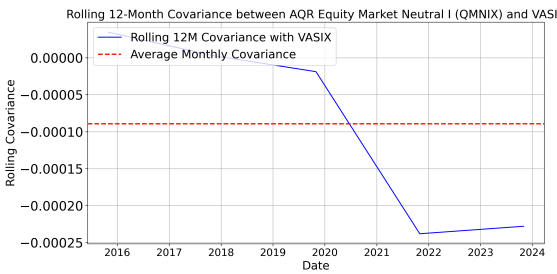

Rolling 12-Month Covariance between iShares Gold Trust (IAU) and VASIX
Rolling 12M Covariance with VASIX
Average Monthly Covariance
Rolling Covariance
Date

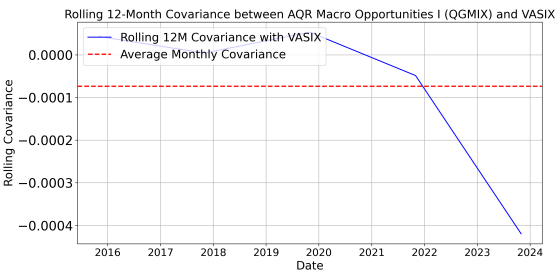

Rolling 12-Month Covariance between Bitcoin Market Price USD (^BTC) and VASIX
Rolling 12M Covariance with VASIX
Average Monthly Covariance
Rolling Covariance
Date

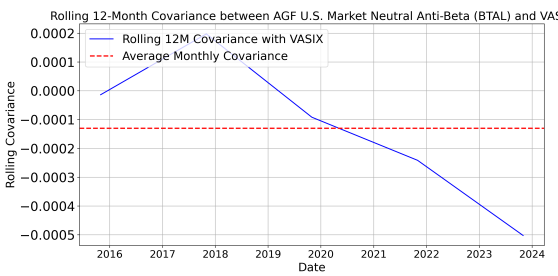

Rolling 12-Month Covariance between AQR Risk-Balanced Commodities Strategy I (ARCIX) an
Rolling 12M Covariance with VASIX
Average Monthly Covariance
Rolling Covariance
Date

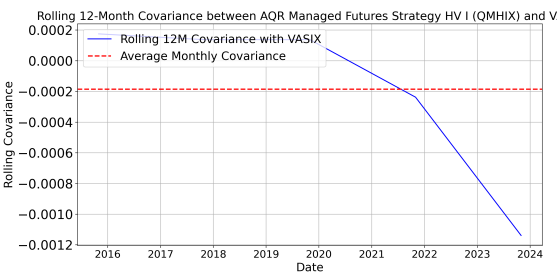

Rolling 12-Month Covariance between AQR Long-Short Equity I (QLEIX) and VASIX
Rolling 12M Covariance with VASIX
Average Monthly Covariance
Rolling Covariance
Date

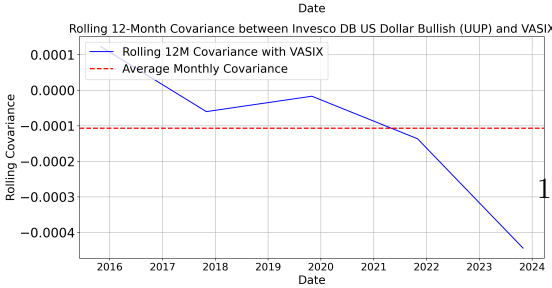

Rolling 12-Month Covariance between AQR Style Premia Alternative I (QSPIX) and VAS
Rolling 12M Covariance with VASIX
Average Monthly Covariance
Rolling Covariance
Date

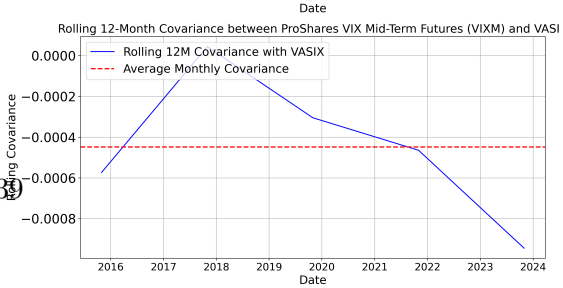

Rolling 12-Month Covariance between AQR Equity Market Neutral I (QMNIX) and VASI
Rolling 12M Covariance with VASIX
Average Monthly Covariance
Rolling Covariance
Date


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


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


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


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


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)

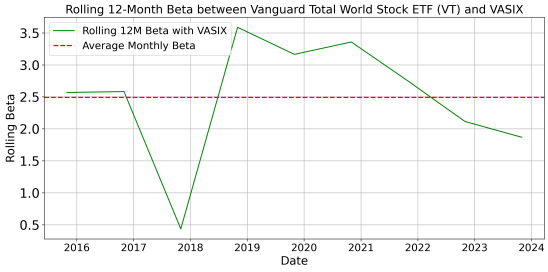
[33]:
```
import os
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from PIL import Image
from IPython.display import display
from scipy import stats

# ------------------------------ #
#         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()
```

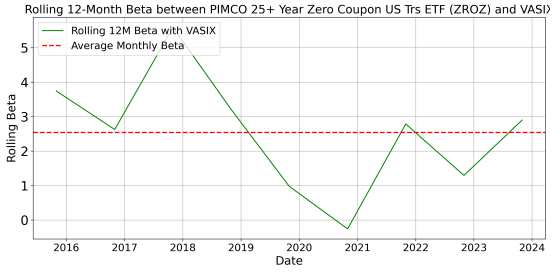
```python
# ------------------------------ #
#      Prepare Output Folder     #
# ------------------------------ #

# 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]
```

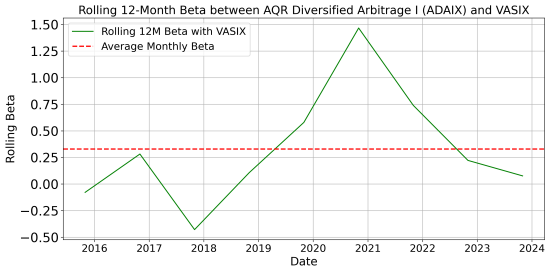
```
# Initialize a dictionary to store rolling betas
rolling_betas = pd.DataFrame()

# 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}')
```

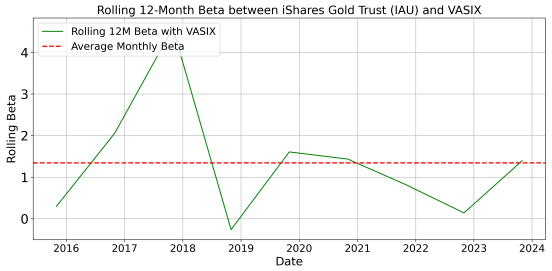
```
    # Calculate and plot the average monthly beta over the full timespan
    avg_beta = beta_series.mean()
    plt.axhline(y=avg_beta, color='red', linestyle='--', linewidth=2,␣
 ↪label='Average Monthly Beta')

    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]
```

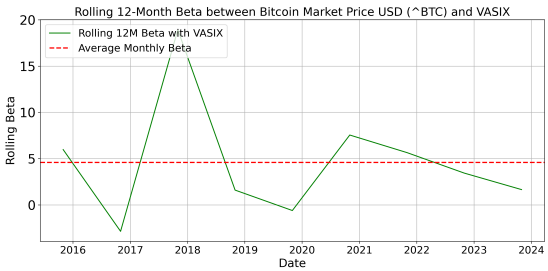
```
# Verify that plot files exist
existing_plot_files = [os.path.join(output_folder, fname) for fname in␣
 ↪plot_filenames if os.path.exists(os.path.join(output_folder, fname))]

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)
```

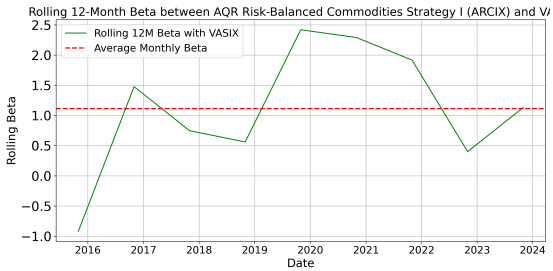

Rolling 12-Month Beta between Vanguard Total World Stock ETF (VT) and VASIX
Rolling 12M Beta with VASIX
Average Monthly Beta
Rolling Beta
Date

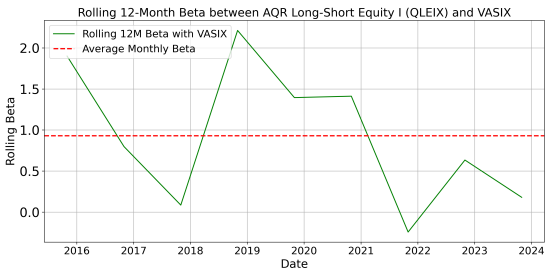

Rolling 12-Month Beta between PIMCO 25+ Year Zero Coupon US Trs ETF (ZROZ) and VASI
Rolling 12M Beta with VASIX
Average Monthly Beta
Rolling Beta
Date

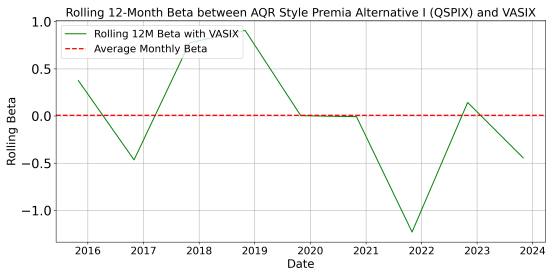

Rolling 12-Month Beta between AQR Diversified Arbitrage I (ADAIX) and VASIX
Rolling 12M Beta with VASIX
Average Monthly Beta
Rolling Beta
Date

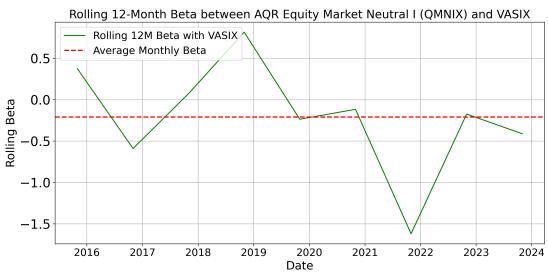

Rolling 12-Month Beta between iShares Gold Trust (IAU) and VASIX
Rolling 12M Beta with VASIX
Average Monthly Beta
Rolling Beta
Date

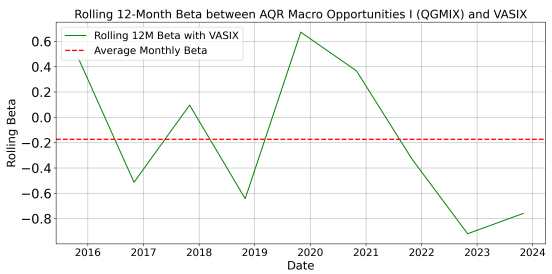

Rolling 12-Month Beta between Bitcoin Market Price USD (^BTC) and VASIX
Rolling 12M Beta with VASIX
Average Monthly Beta
Rolling Beta
Date

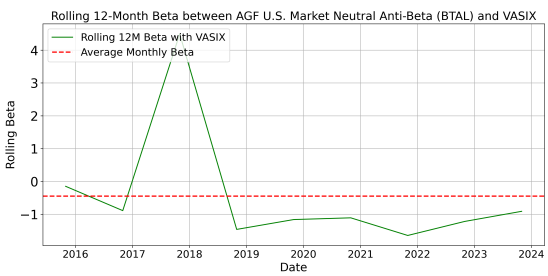

Rolling 12-Month Beta between AQR Risk-Balanced Commodities Strategy I (ARCIX) and VA
Rolling 12M Beta with VASIX
Average Monthly Beta
Rolling Beta
Date

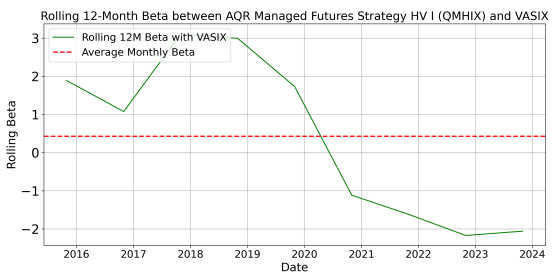

Rolling 12-Month Beta between AQR Long-Short Equity I (QLEIX) and VASIX
Rolling 12M Beta with VASIX
Average Monthly Beta
Rolling Beta
Date

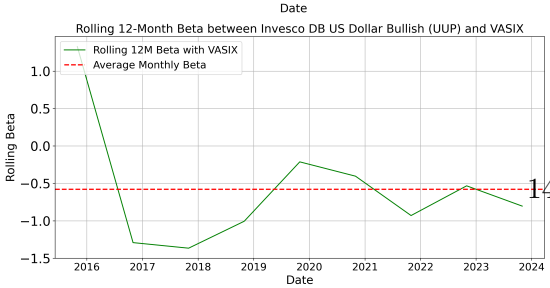

Rolling 12-Month Beta between AQR Style Premia Alternative I (QSPIX) and VASIX
Rolling 12M Beta with VASIX
Average Monthly Beta
Rolling Beta
Date

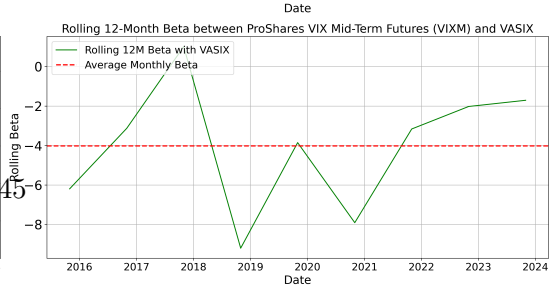

Rolling 12-Month Beta between AQR Equity Market Neutral I (QMNIX) and VASIX
Rolling 12M Beta with VASIX
Average Monthly Beta
Rolling Beta
Date


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:
```

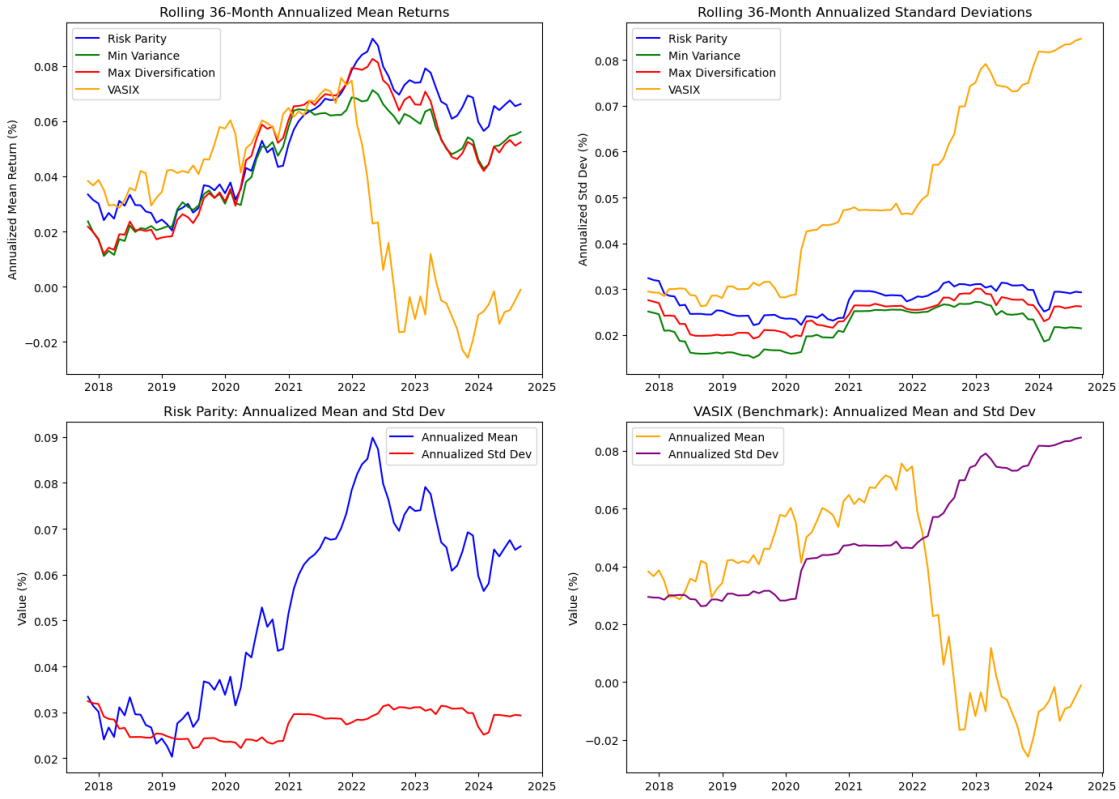
```
    regression_output_folder = 'REGRESSION_RESULTS'
    os.makedirs(regression_output_folder, exist_ok=True)
    regression_output_file_path = os.path.join(regression_output_folder,␣
 ↪"regression_results_table.png")
    try:
        dfi.export(styled_regression_table, regression_output_file_path)
    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}")
```



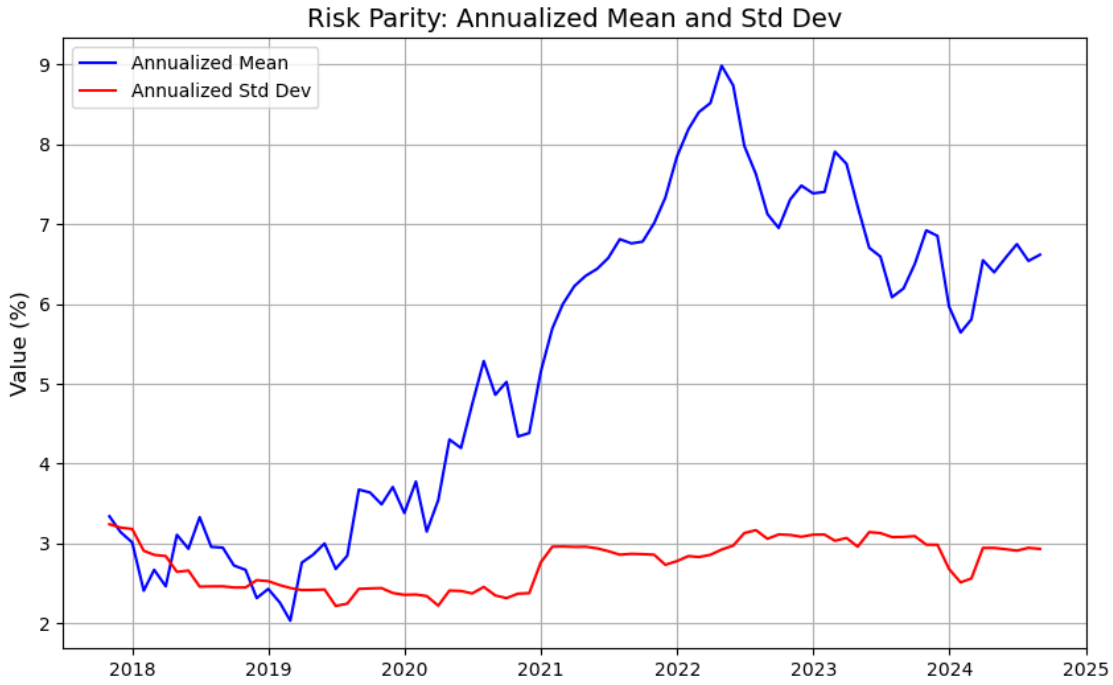

```
Plot saved as rolling_36_month_mean_std_plot.png
```

31 Individual Optimized Portfolio Rolling 36-month Mean and Standard Deviations (Not Shown in Report)

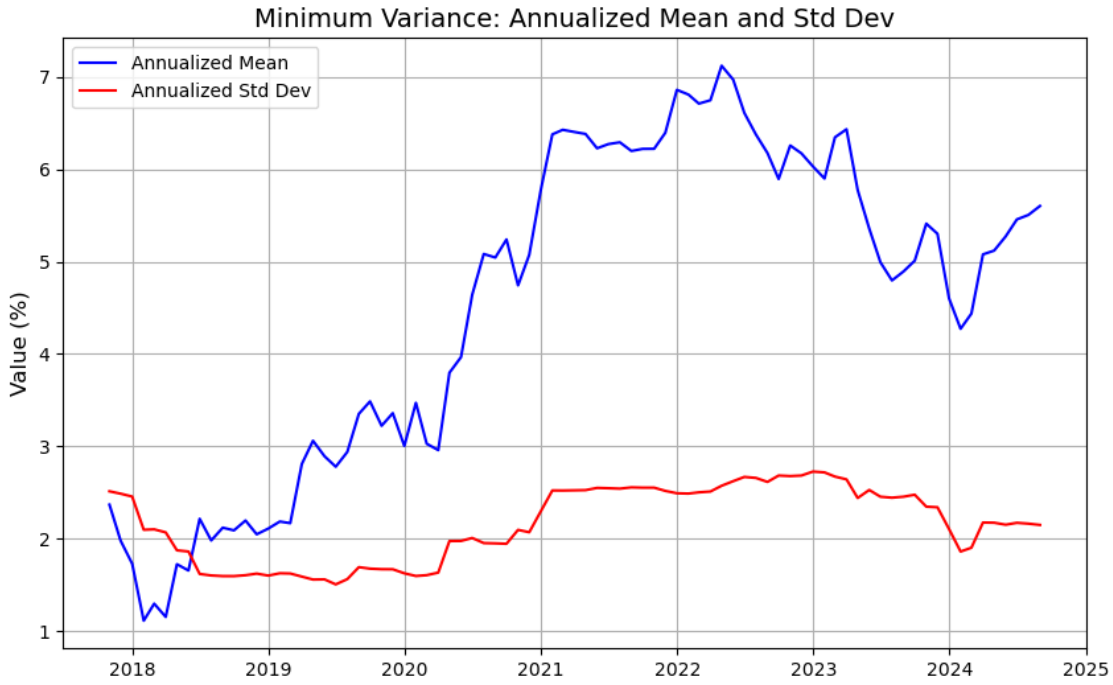
[50]:
```
import matplotlib.pyplot as plt
import os

# 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
```

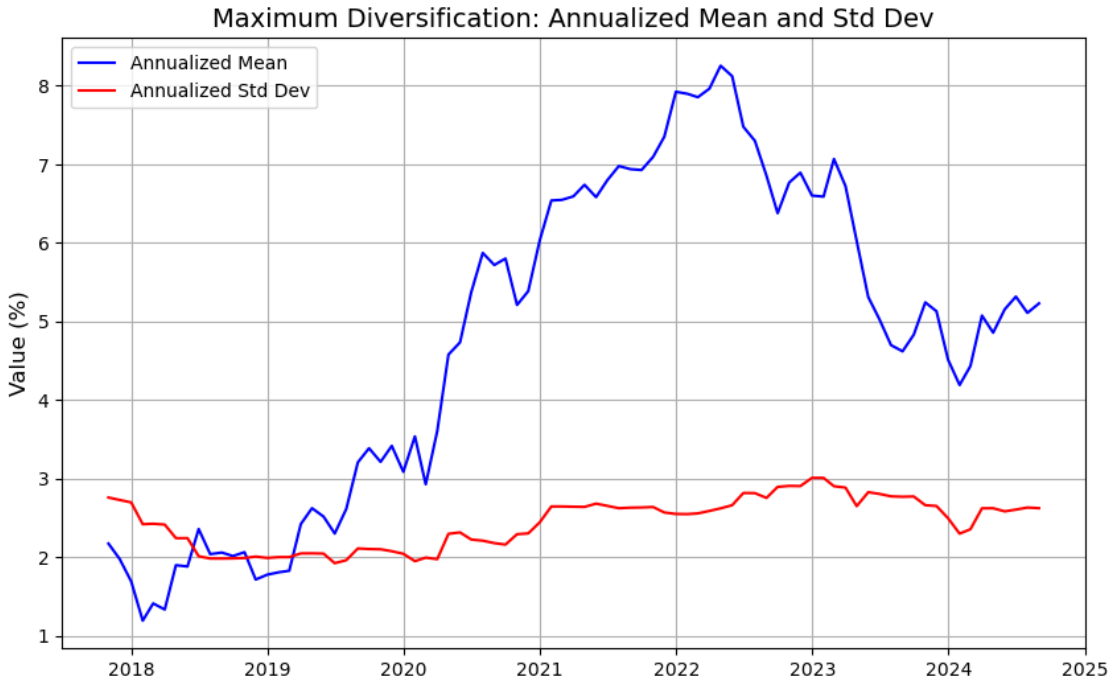
```
})

# 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}")
```



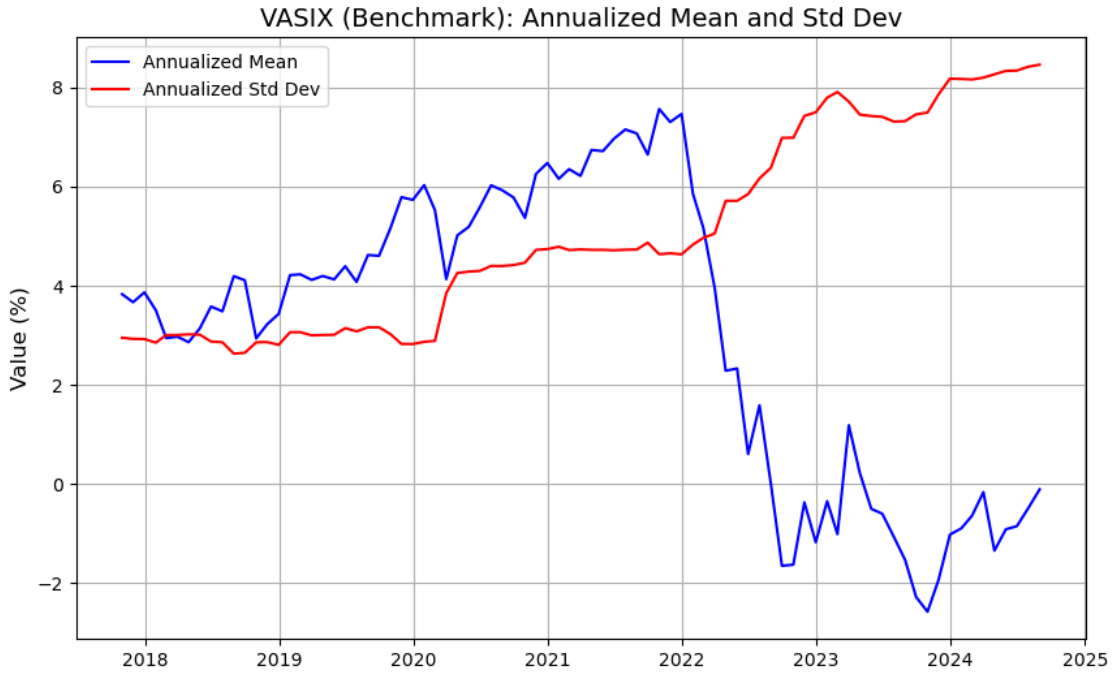
Individual plot saved as Rolling_Stats_Plots/risk_parity_rolling_stats.png



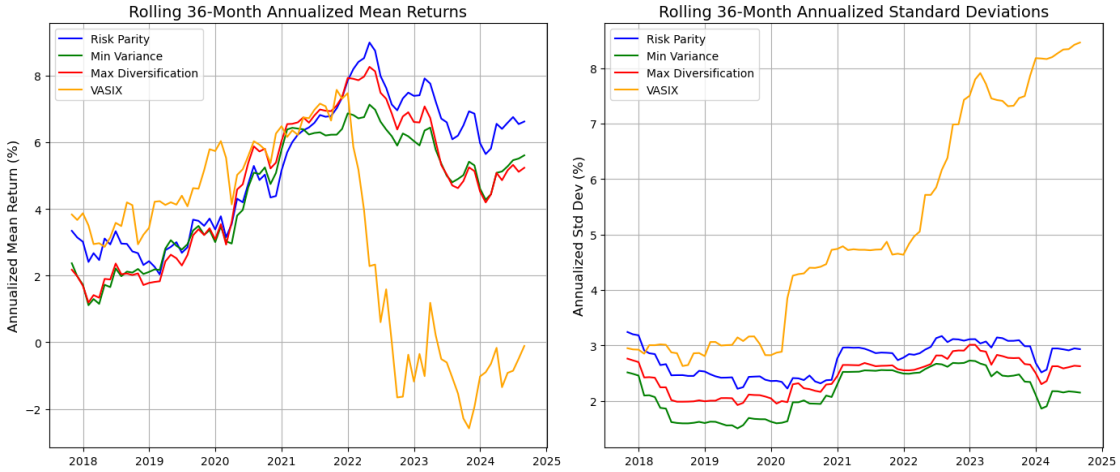
Individual plot saved as Rolling_Stats_Plots/minimum_variance_rolling_stats.png




```
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]:
```python
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>

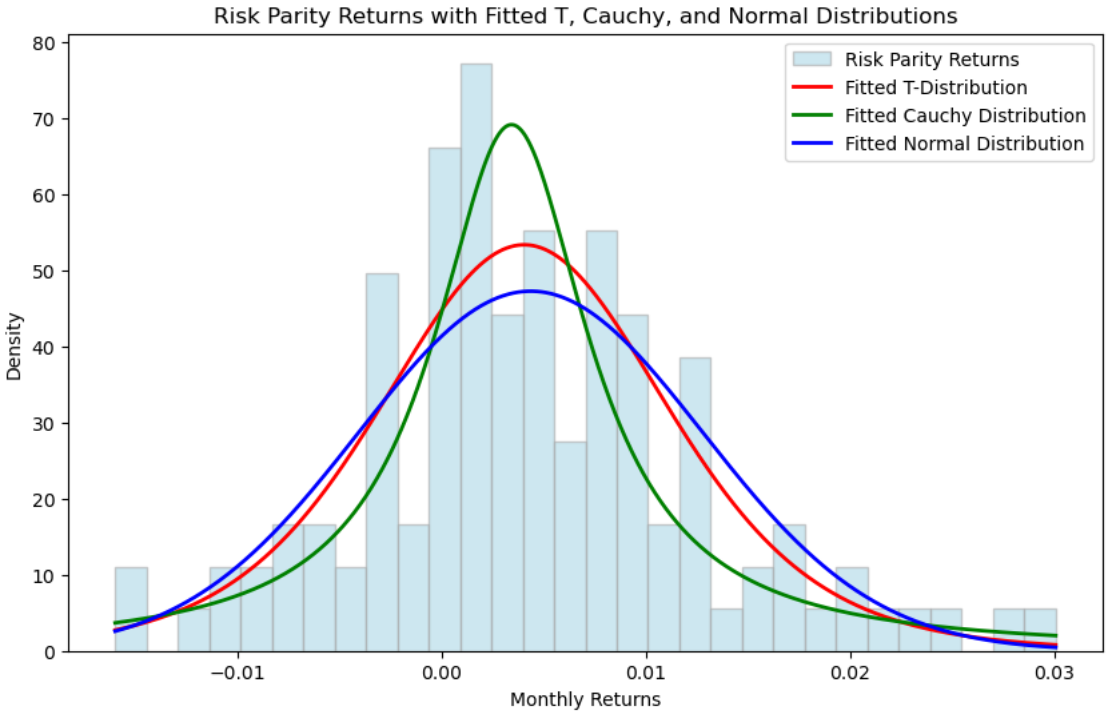
33 Raw Return Histograms of Optimized Portfolios with Fitted Probability Distributions (Not an Exhibit in Report)

[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)'
```

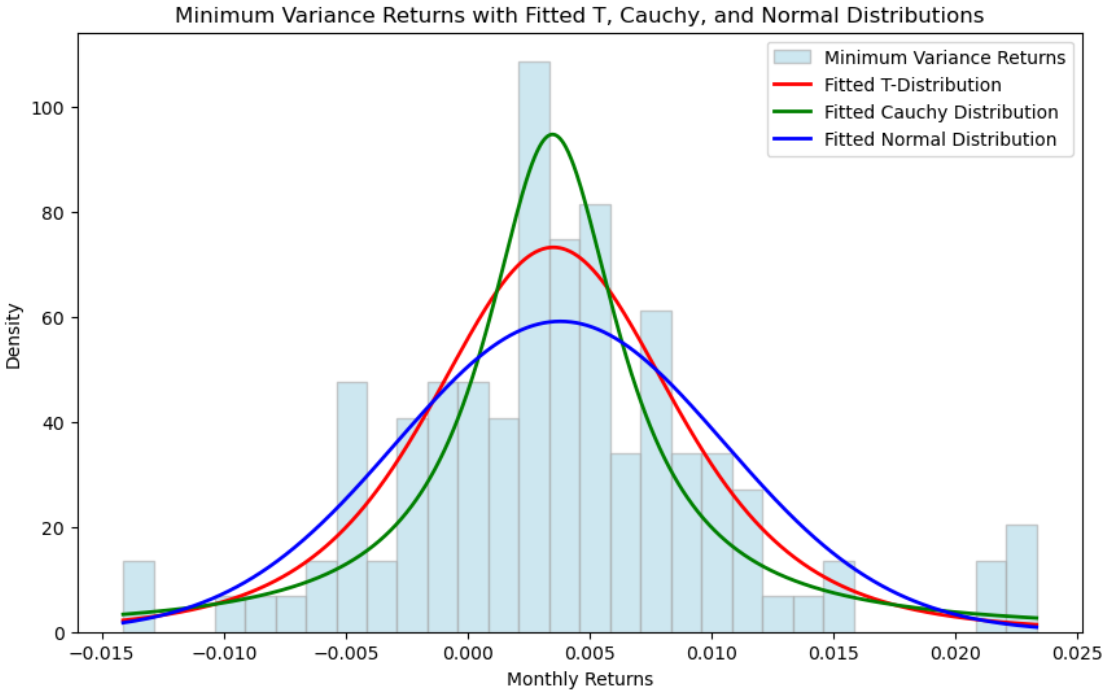
```
# List of portfolio names
portfolios = ['Risk Parity', 'Minimum Variance', 'Maximum Diversification']

# 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)
```

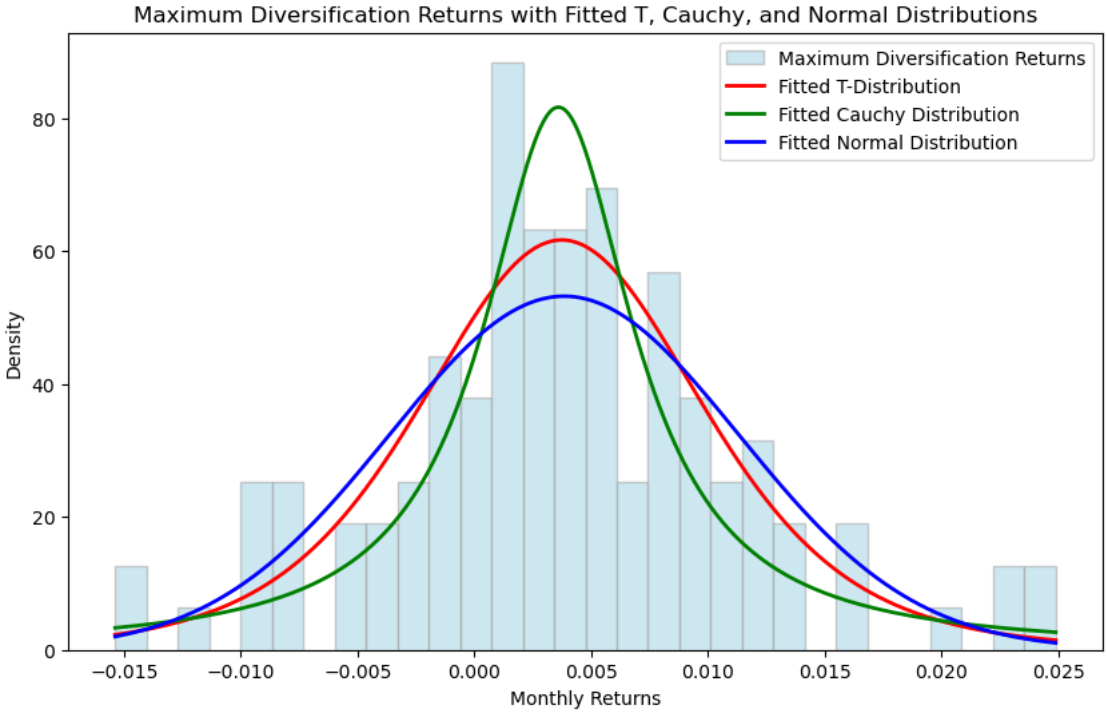
```
Generating plot for Risk Parity…
```




```
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
```

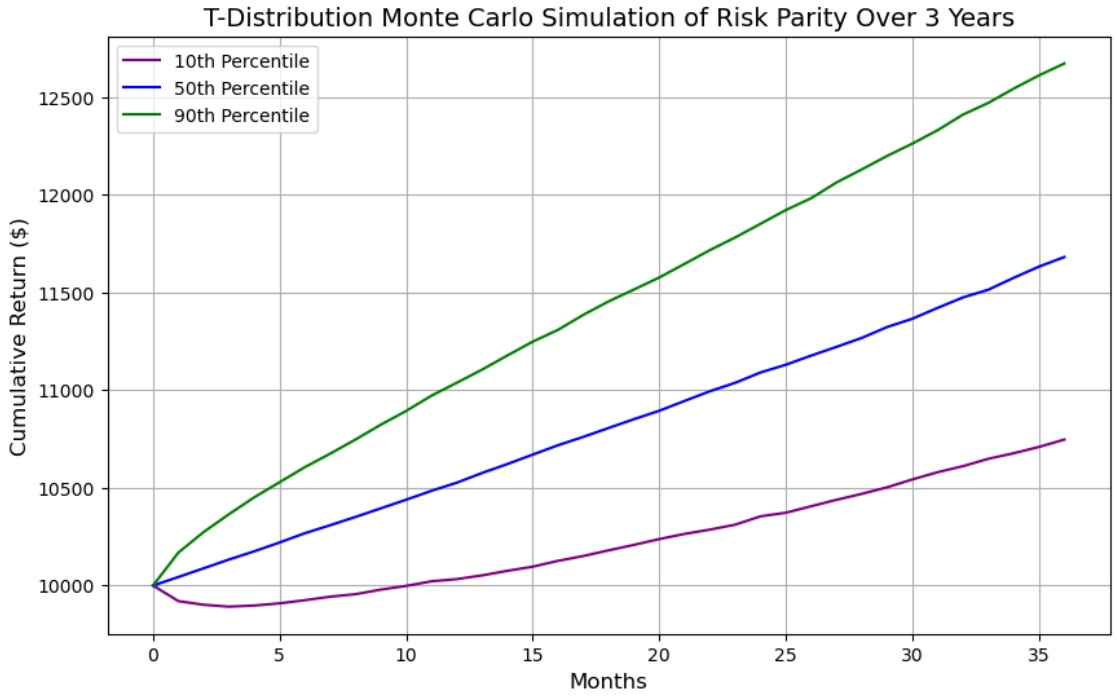
```
    simulations = t.rvs(df, loc=mean_monthly, scale=std_monthly,␣
↪size=(num_simulations, months))

    # 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))
```

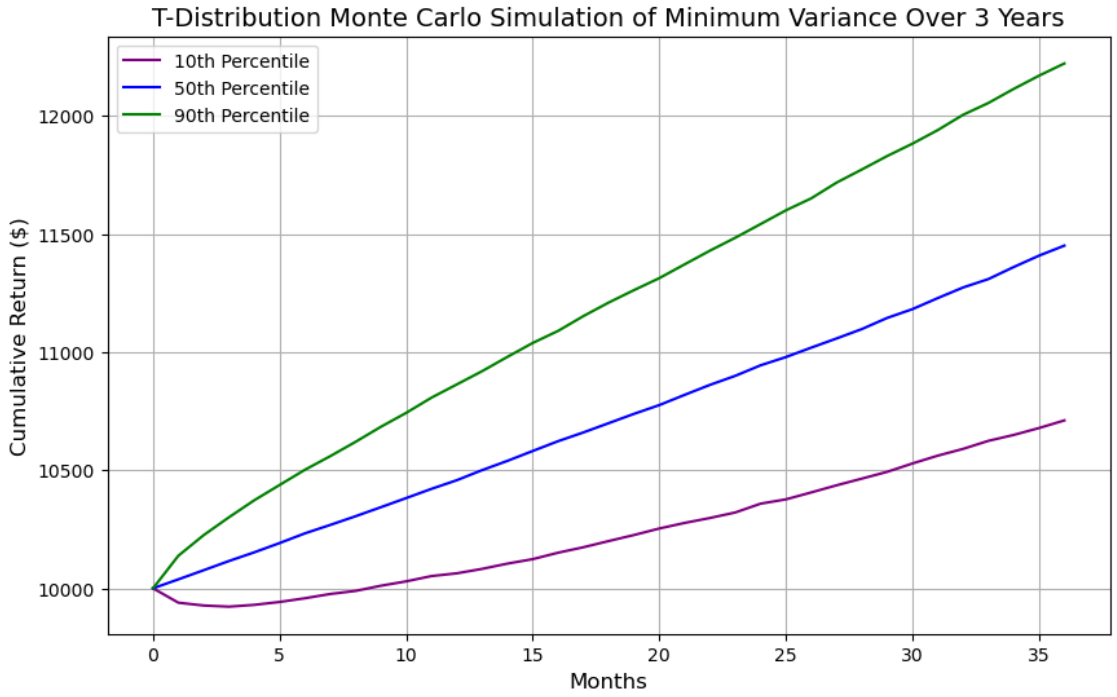
```
    max_width = max(widths)
    max_height = max(heights)

    # 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
```



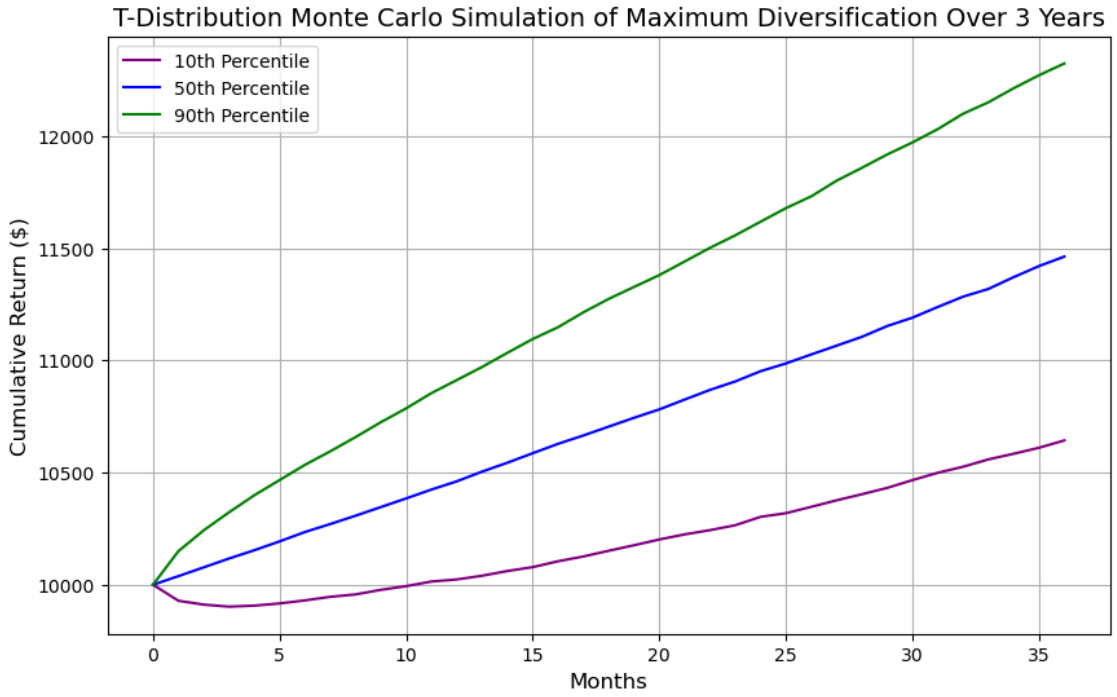
	Portfolio	Percentile	Portfolio End Balance ($)	Annual Compounded Return (%)	Annualized Volatility (%)	Maximum Drawdown (%)
0	Risk Parity	90th Percentile	12,672	8.21	4.54	-1.44
1	Risk Parity	50th Percentile	11,681	5.32	3.58	-2.74
2	Risk Parity	10th Percentile	10,747	2.43	2.89	-5.38

Running Rolling Monte Carlo Simulation for Minimum Variance



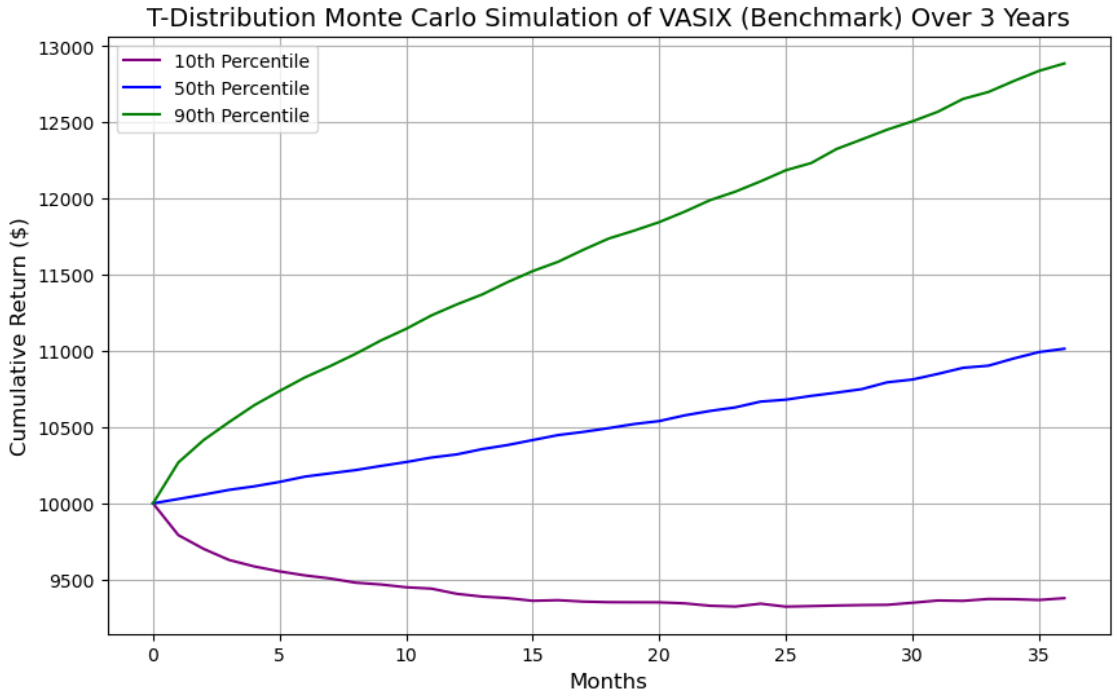
	Portfolio	Percentile	Portfolio End Balance ($)	Annual Compounded Return (%)	Annualized Volatility (%)	Maximum Drawdown (%)
0	Minimum Variance	90th Percentile	12,222	6.92	3.63	-1.09
1	Minimum Variance	50th Percentile	11,451	4.62	2.87	-2.09
2	Minimum Variance	10th Percentile	10,711	2.32	2.31	-4.13

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

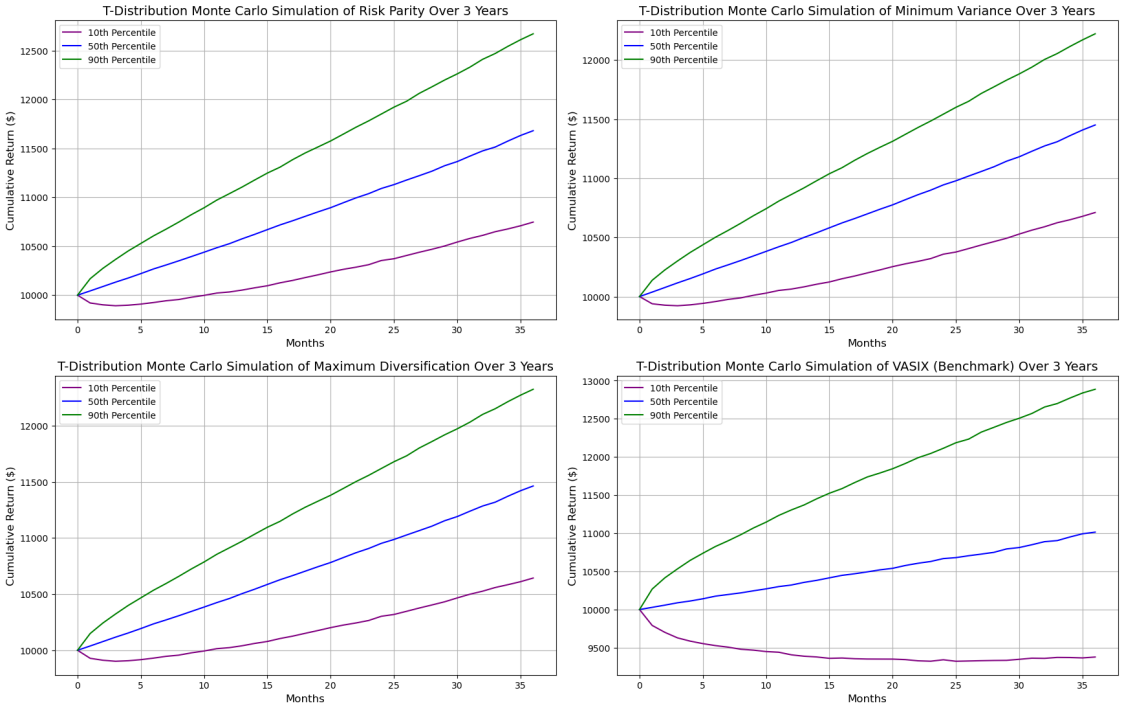
Running Rolling Monte Carlo Simulation for VASIX (Benchmark)



	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