Modern Portfolio Theory - S&P 500 - Daily Rebalance Period

July 24, 2023

1 Abstract

The Modern Portfolio Theory (MPT) also known as the Markowitz model, is an portfolio optimisation technique, which aims to maximise expected returns whilst considering the potential risk.

This script builds on from the crypto MPT we produced. This portfolio is applied to equities, considering 3 variaitons of the MPT. Each of the 3 portfolios considered here result in insignificant returns. MPT is not a strategy which we should continue pursuing.

2 Introduction

The portfolios in this test applies to a basket of equities, made up of the individual securities which make up the S&P 500. With a daily rebalance period. Meaning each day, the weights applied to each of the equities are recalculated, based on the latest information including: changes in covariance between equity pairs, the latest returns based on the most recent day is also now being considered. The weightings are determined by maximising the sharpe ratio (in this case), which considers the returns and the volatility based on historical data. The various portfolio weightings and their corresponding expected returns and volatiltiy can be visualised on a scatter plot, where the efficient frontier is also present. This is a plot which highlights the various portfolios which achieve a certain return but with the lowest risk associated. Hence the efficient frontier lies to the left side of the main plot, further left means lower expected volatility, as volatility is measured on the x-axis.

The weigths for the lowest volatility can be solved for by taking the portfolio variance's partial derivative with respect to a single weight, then rearranging for the weight, this is simply modelled with a portfolio of just 2 securities. The process does not work by producing many random weighted portfolios, then chosing the best based on the clients requirements, even though it can be solved via this route, it is not efficient and may not result in an optimal solution. Portfolio variance formula for 2 securities:

$$(\sigma_{portfolio})^2 = (1-w)^2\sigma_1^2 + w^2\sigma_2^2 + 0$$

Where: w is the weight for security 2. $(1-w)$ is the weight for security 1. σ_1 is the standard deivation of security 1 (known). σ_2 is the standard deviation of security 2 (known). And the 0 represents no correlation between the 2 securities, this value can change depending on the covariance between the securities.

Take the partial derivative with respect to the weight, and then set equal to zero, to find the minimum variance portfolio weightings:

$$\frac{\partial(\sigma_{portfolio})^2}{\partial w} = -2(1-w)\sigma_1^2 + 2w\sigma_2^2$$

Rearrange for w:

$$w = \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2}$$

This is the weight associated with security 2, for achieving a portfolio with the lowest volatiltiy, represented by the furthest left point on an efficient frontier. Weight 2 can then be sovled from this. This will allow us to remove all the sub-optimal portfolios (portfolios below this location on the y-axis are considered sub-optimal). From the formula for w, the weights are inversely proportional to their squared volatility, remember w is the weight associated with security 2.

Model assumptions: Ultra low latency. Zero slippage. Only accounts for revenue, no bid-ask spread or commission fees have been considered yet. For trading fees, consider: Cloud server costs (per hour or usage), API connection fee (for datastream and our API provider to act as a brokerage), we are trading 500 equities, every day, this will results in high costs.

Updates to this Notebook This notebook now has 4 main parts within the results section. **Part 1** is associated with MPT where the mean-variance tradeoff issue is present, which is a tradeoff between expected returns and risk. This results in weightings of certain assets being set to 0. So really, the portfolio is less diversified. Additionally, the covariance matrix is calculated using the sample covariance method.

Part 2 is the MPT with the mean-variance tradeoff issue present. And now the covariance matrix is attempted to be solved for by using the Ledoit-Wolf shrinkage, which is a regularisation technique which helps provide stability to the matrix.

Part 3 is the MPT with the mean-variance tradeoff issue removed via the incorporation of L2 regularisation. And And now the covariance matrix is attempted to be solved for by using the Ledoit-Wolf shrinkage, which is a regularisation technique which helps provide stability to the matrix.

Part 4 is a regular buy and hold strategy of the S&P 500 index fund. The equities we consider in this MPT are sourced from the S&P 500 companies, therefore our MPT must atleast be able to outpreform its underlying security: the S&P 500.

(The Ledoit-wolf shrinkage technique is an iterative process and is more computationally expensive in comparison to the sample covariance solver method, but it has been considered here in an attempt to avoid the covariance method being non-positive semi definite or non positive definite).

3 Futher MPT explanation

The above section provided the solution for identifying the portfolio weights which achieve a portfolio with the lowest portfolio variance, which helps identify all sub-optimal portfolios. Now we may have further constraints, such as we may want to define an expected return we would ideally like to achieve, besides minimising the portfolio variance.

This is a quadratic programming problem, specifically where we want to minimise the squared volatility of our portfolio subject to a constraint mean return and considering that we are fully invested. To solve this issue, we need to define a Lagrangian. We are going to take the objective function of the volatility which we want to minimise, and then add lagrangians for the different constraints.

The Lagrangian function is formulated as follows:

$$\mathcal{L}(\mathbf{w}, \sigma_1, \sigma_2) = \frac{1}{2}\mathbf{w}'\Sigma\mathbf{w} + \lambda_1(\alpha_0 - \mathbf{w}'\mathbf{R}) + \lambda_2(1 - \mathbf{w}'\mathbf{1_m})$$

Where,

$\mathbf{w}$ is the vector of portfolio weights.

Σ is the covariance matrix of asset returns.

α_0 is the target expected return for the portfolio.

$\mathbf{R}$ is the vector of expected returns for individual assets.

$\mathbf{1_m}$ is an indicator function (a vector with elements 0 or 1, where 1 indicates that the corresponding asset is included in the portfolio and 0 indicates exclusion).

$\mathbf{w}'$ represents the transpose of vector $\mathbf{w}$.

To find the optimal portfolio weights, we need to derive the first-order conditions with respect to $\mathbf{w}$ and set the derivatives equal to zero:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \Sigma\mathbf{w} - \lambda_1\mathbf{R} - \lambda_2\mathbf{1_m} = 0$$

Solve this equation for $\mathbf{w}$ in terms of λ_1 and λ_2:

$$\mathbf{w} = \Sigma^{-1}(\lambda_1\mathbf{R} + \lambda_2\mathbf{1_m})$$

Next, we consider the portfolio constraint equations:

$\frac{\partial \mathcal{L}}{\partial \lambda_1} = 0 = \alpha_0 - \mathbf{w}'\mathbf{R} = 0$ (Constraint 1)

$\frac{\partial \mathcal{L}}{\partial \lambda_2} = 0 = 1 - \mathbf{w}'\mathbf{1_m}$ (Constraint 2)

Substitute the expression for $\mathbf{w}$ from the first-order condition into these constraint equations:

$\alpha_0 - (\Sigma^{-1}(\lambda_1\mathbf{R} + \lambda_2\mathbf{1_m}))'\mathbf{R} = 0$ (Constraint 1)

$\mathbf{1}'_\mathbf{m}(\Sigma^{-1}(\lambda_1\mathbf{R} + \lambda_2\mathbf{1_m})) - 1 = 0$ (Constraint 2)

Now, we have two equations with two unknowns (λ_1 and λ_2). Solve these equations simultaneously to find the values of λ_1 and λ_2.

Finally, substitute the values of λ_1 and λ_2 back into the expression for $\mathbf{w}$ to get the optimal portfolio weights.

This will be discussed below, but occasionally solving for a covariance matrix may result in it being non-positive semi definite, where eigenvalues are zero or negative. A non-positive semi definite

covariance matrix can cause problems in the optimisation process, leading to sub-optimal or even invalid portfolio allocations, largely due to issues with the inverse covariance matrix.

To overcome this issue, I will test the incorporation of the Ledoit-Wolf Shrinkage regularisation technique, where the results are produced in **Part 2** and **Part 3**.

4 Issues of this equity MPT

Our original crypto MPT, we worked with 12 cryptocurrencies, and calculated their corresponding covariance matricies using 30 observations in a loop which modelled a moving window with a period=30. Meaning the number of features (p) here was 12, and the number of observations (N) is 30. So $p < N$. In this new situtation where we are now applying a MPT to 500 equities, the same 500 equities which are present in the S&P 500, and if we keep the number of observations to 30 ($N = 30$ and $p = 500$), we will now encounter issues related to high dimensionality. Where $p >> N$. This is also known as the curse of dimensionality. A primary issue where we have 500 features and 30 observations to calculate a covariance matrix, is that the number of observations is not sufficient to capture any meaningful relationships, other issues include: overfitting, increased noise sensitivity, increased computational complexity (as the number of features increases the computational complexity of algorithms and models increases exponentially). Primary starting solution is simply to increase our number of observations from 30, note the length of our dataframe which contains the total number of observations available is currently 884 rows representing trading days in the range (2019/11/22 - 2023/05/31) will decrease as our number of observations increases.

Now when calculating the best portfolio using MPT, the **pypfopt** libary encounters an important concern when attempting to solve the covariance matricies. Certain covariance maticies have the issue of being non positive semidefinite (atleast one of the eigenvalues are negative), whereas a covariance matrix should be positive semidefinite, meaning all the eigenvalues are positive.

However, due to noise, errors, or insufficient data, the estimated covariance matrix can become ill-conditioned, which means some of its eigenvalues are negative or very close to zero. This situation is problematic because a non-positive semidefinite covariance matrix is not mathematically valid and can cause issues in portfolio optimization, risk management, and other statistical analyses.

This code encounters issues of the covariance matricies being non positive semidefinite, in such cases, the **pypfopt** library attempts to adjust the matrix using various techniques, such as regularization, shrinkage, or eigenvalue correction. Even then, the pypfopt libary may occasionally fail to fix (solve) this issue using alternative methods of finding the covariance matrix. I will proceed to use a covariance estimator: Ledoit-Wolf shrinkage. Covariance estimators do not directly solve for the true population covariance matrix, but rather attempt to provide an estimate that is as close as possible to the true covariance matrix based on the available data. Covariance estimators use numerical estimation process rather than a direct mathematical solution. Ledoit-Wolf shrinkage introduces bias to stabilise the covariance matrix estimates when dealing with high-dimensional data or limited samples, where we orginally had 500 features and 30 observations ($p >> N$).

The Ledoit-Wolf shrinkage applied in the context of estimating the covariance matrix, the sample covariance matrix is calculated directly from the observed data, but it can be highly sensitive to noise and outliers, especially when the number of observations is limited. Ledoit-Wolf shrinkage addresses this issue by borrowing information from the sample covariance matrix (the empirical estimate of the covariance matrix) and shrinks the off-diagonal elements towards the diagonal elements of the sample covariance matrix.

Additionally, our script captures the current S&P 500 equities. Some of these equities will not have been present in the S&P 500 on the 22-11-2019. This is an issue, as considering this equity before it's incorporation is likely to result in slighlty incorrect results: as their market capitalisation future growth is going to be significant if it will eventaully join the S&P 500 (which it now has). Therefore, this code has selected a company which eventually joins the S&P 500, which is actually very hard to predict in real time. Another issue here is some companies may not have even been public (22-11-2019) resulting in certain cells of our dataframe containing 'NaN', which can result in errors or invalid solutions when calculating the necessary parts for solving the mean-variance optimisation (MVO) solver. Therefore these companies have been removed from the test. The number of companies is minimal and our test works with >490 equities.

Other small considerations have to be made including, when calculting the returns via pct_change(), row index[0] will now contain 'nan'. Therefore removing row index[0] from the returns dataframe is necessary to avoid issues when calculating the mean expected return. Now, returns.index[0] corresponds to the next point in time, whilst df.index[0] corresponds to the current point in time. Hence, when using 'returns' there is a look-ahead bias present. Even though in this specific code, the issue will result in negligable concerns, these concerns still exist. Therefore removing df.index[0] is to be removed. Additionally there is an issue experienced when considered short sells through the **pypfopt** package. For a portfolio of weights $\mathbf{w}$ the $\sum_{i=1}^{n} w_i = 1$. However if short selling is enabled, the **pypfopt** package can have weightings of [0.2,0.5,0.6,0.3,-0.8,-0.8], here, $\sum_{i=1}^{n} w_i = 1$ still. Even though this is not possible, the **pypfopt** has a constraint so the $\sum_{i=1}^{n} w_i$ still equal to 1, this results in unrealistic portfolio weighting allocations, as the long investments $\sum_{w>0} w > 1$, this is not possible as a value greater than 1, represents captial we don't have access to, the value of 1.0, represents all our capital available.

5 Importing relevant packages

[18]:
```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from pypfopt import expected_returns, risk_models, objective_functions
from pypfopt.efficient_frontier import EfficientFrontier
import warnings
warnings.filterwarnings("ignore", category=UserWarning, message="max_sharpe␣
 ↪transforms the optimization problem")
import finnhub
import requests
from bs4 import BeautifulSoup
from sklearn.covariance import LedoitWolf
import cvxpy as cp
from cvxpy import ECOS, SCS
```

6 Importing the tickers and data

Firstly we need to identify the tickers which make up the S&P 500, so we can then proceed with extracting their individual adjusted close data. Additionally, we will extracted the points of the

S&P 500 over this equivalent period. As our MPT is made up of the S&P 500 assets, our goal is to outpreform the S&P 500 returns, based on point change over this period of time (22-11-2019 to 31-05-2023).

[19]:
```
url = 'https://en.wikipedia.org/w/index.php?title=List_of_S%26P_500_companies'
response = requests.get(url)
soup = BeautifulSoup(response.content, 'html.parser')
table = soup.find('table', {'id': 'constituents'})
tickers = []
for row in table.find_all('tr')[1:]:
    columns = row.find_all('td')
    ticker = columns[0].text.strip()
    tickers.append(ticker)
print(tickers)

df = yf.download(tickers, start="2019-11-22", end="2023-05-31")['Adj Close']
df = df.dropna(axis=1)

print(df.head())

spy_data=yf.download("SPY", start="2019-11-22", end="2023-05-31")['Adj Close']
```

```
['MMM', 'AOS', 'ABT', 'ABBV', 'ACN', 'ATVI', 'ADM', 'ADBE', 'ADP', 'AAP', 'AES',
'AFL', 'A', 'APD', 'AKAM', 'ALK', 'ALB', 'ARE', 'ALGN', 'ALLE', 'LNT', 'ALL',
'GOOGL', 'GOOG', 'MO', 'AMZN', 'AMCR', 'AMD', 'AEE', 'AAL', 'AEP', 'AXP', 'AIG',
'AMT', 'AWK', 'AMP', 'ABC', 'AME', 'AMGN', 'APH', 'ADI', 'ANSS', 'AON', 'APA',
'AAPL', 'AMAT', 'APTV', 'ACGL', 'ANET', 'AJG', 'AIZ', 'T', 'ATO', 'ADSK', 'AZO',
'AVB', 'AVY', 'AXON', 'BKR', 'BALL', 'BAC', 'BBWI', 'BAX', 'BDX', 'WRB',
'BRK.B', 'BBY', 'BIO', 'TECH', 'BIIB', 'BLK', 'BK', 'BA', 'BKNG', 'BWA', 'BXP',
'BSX', 'BMY', 'AVGO', 'BR', 'BRO', 'BF.B', 'BG', 'CHRW', 'CDNS', 'CZR', 'CPT',
'CPB', 'COF', 'CAH', 'KMX', 'CCL', 'CARR', 'CTLT', 'CAT', 'CBOE', 'CBRE', 'CDW',
'CE', 'CNC', 'CNP', 'CDAY', 'CF', 'CRL', 'SCHW', 'CHTR', 'CVX', 'CMG', 'CB',
'CHD', 'CI', 'CINF', 'CTAS', 'CSCO', 'C', 'CFG', 'CLX', 'CME', 'CMS', 'KO',
'CTSH', 'CL', 'CMCSA', 'CMA', 'CAG', 'COP', 'ED', 'STZ', 'CEG', 'COO', 'CPRT',
'GLW', 'CTVA', 'CSGP', 'COST', 'CTRA', 'CCI', 'CSX', 'CMI', 'CVS', 'DHI', 'DHR',
'DRI', 'DVA', 'DE', 'DAL', 'XRAY', 'DVN', 'DXCM', 'FANG', 'DLR', 'DFS', 'DIS',
'DG', 'DLTR', 'D', 'DPZ', 'DOV', 'DOW', 'DTE', 'DUK', 'DD', 'DXC', 'EMN', 'ETN',
'EBAY', 'ECL', 'EIX', 'EW', 'EA', 'ELV', 'LLY', 'EMR', 'ENPH', 'ETR', 'EOG',
'EPAM', 'EQT', 'EFX', 'EQIX', 'EQR', 'ESS', 'EL', 'ETSY', 'EG', 'EVRG', 'ES',
'EXC', 'EXPE', 'EXPD', 'EXR', 'XOM', 'FFIV', 'FDS', 'FICO', 'FAST', 'FRT',
'FDX', 'FITB', 'FSLR', 'FE', 'FIS', 'FI', 'FLT', 'FMC', 'F', 'FTNT', 'FTV',
'FOXA', 'FOX', 'BEN', 'FCX', 'GRMN', 'IT', 'GEHC', 'GEN', 'GNRC', 'GD', 'GE',
'GIS', 'GM', 'GPC', 'GILD', 'GL', 'GPN', 'GS', 'HAL', 'HIG', 'HAS', 'HCA',
'PEAK', 'HSIC', 'HSY', 'HES', 'HPE', 'HLT', 'HOLX', 'HD', 'HON', 'HRL', 'HST',
'HWM', 'HPQ', 'HUM', 'HBAN', 'HII', 'IBM', 'IEX', 'IDXX', 'ITW', 'ILMN', 'INCY',
'IR', 'PODD', 'INTC', 'ICE', 'IFF', 'IP', 'IPG', 'INTU', 'ISRG', 'IVZ', 'INVH',
'IQV', 'IRM', 'JBHT', 'JKHY', 'J', 'JNJ', 'JCI', 'JPM', 'JNPR', 'K', 'KDP',
'KEY', 'KEYS', 'KMB', 'KIM', 'KMI', 'KLAC', 'KHC', 'KR', 'LHX', 'LH', 'LRCX',
```

```
'LW', 'LVS', 'LDOS', 'LEN', 'LNC', 'LIN', 'LYV', 'LKQ', 'LMT', 'L', 'LOW',
'LYB', 'MTB', 'MRO', 'MPC', 'MKTX', 'MAR', 'MMC', 'MLM', 'MAS', 'MA', 'MTCH',
'MKC', 'MCD', 'MCK', 'MDT', 'MRK', 'META', 'MET', 'MTD', 'MGM', 'MCHP', 'MU',
'MSFT', 'MAA', 'MRNA', 'MHK', 'MOH', 'TAP', 'MDLZ', 'MPWR', 'MNST', 'MCO', 'MS',
'MOS', 'MSI', 'MSCI', 'NDAQ', 'NTAP', 'NFLX', 'NWL', 'NEM', 'NWSA', 'NWS',
'NEE', 'NKE', 'NI', 'NDSN', 'NSC', 'NTRS', 'NOC', 'NCLH', 'NRG', 'NUE', 'NVDA',
'NVR', 'NXPI', 'ORLY', 'OXY', 'ODFL', 'OMC', 'ON', 'OKE', 'ORCL', 'OGN', 'OTIS',
'PCAR', 'PKG', 'PANW', 'PARA', 'PH', 'PAYX', 'PAYC', 'PYPL', 'PNR', 'PEP',
'PFE', 'PCG', 'PM', 'PSX', 'PNW', 'PXD', 'PNC', 'POOL', 'PPG', 'PPL', 'PFG',
'PG', 'PGR', 'PLD', 'PRU', 'PEG', 'PTC', 'PSA', 'PHM', 'QRVO', 'PWR', 'QCOM',
'DGX', 'RL', 'RJF', 'RTX', 'O', 'REG', 'REGN', 'RF', 'RSG', 'RMD', 'RVTY',
'RHI', 'ROK', 'ROL', 'ROP', 'ROST', 'RCL', 'SPGI', 'CRM', 'SBAC', 'SLB', 'STX',
'SEE', 'SRE', 'NOW', 'SHW', 'SPG', 'SWKS', 'SJM', 'SNA', 'SEDG', 'SO', 'LUV',
'SWK', 'SBUX', 'STT', 'STLD', 'STE', 'SYK', 'SYF', 'SNPS', 'SYY', 'TMUS',
'TROW', 'TTWO', 'TPR', 'TRGP', 'TGT', 'TEL', 'TDY', 'TFX', 'TER', 'TSLA', 'TXN',
'TXT', 'TMO', 'TJX', 'TSCO', 'TT', 'TDG', 'TRV', 'TRMB', 'TFC', 'TYL', 'TSN',
'USB', 'UDR', 'ULTA', 'UNP', 'UAL', 'UPS', 'URI', 'UNH', 'UHS', 'VLO', 'VTR',
'VRSN', 'VRSK', 'VZ', 'VRTX', 'VFC', 'VTRS', 'VICI', 'V', 'VMC', 'WAB', 'WBA',
'WMT', 'WBD', 'WM', 'WAT', 'WEC', 'WFC', 'WELL', 'WST', 'WDC', 'WRK', 'WY',
'WHR', 'WMB', 'WTW', 'GWW', 'WYNN', 'XEL', 'XYL', 'YUM', 'ZBRA', 'ZBH', 'ZION',
'ZTS']
[**********************100%***********************]  503 of 503 completed

2 Failed downloads:
- BF.B: No data found for this date range, symbol may be delisted
- BRK.B: No timezone found, symbol may be delisted
                                A         AAL         AAP        AAPL        ABBV  \
Date
2019-11-22 00:00:00  77.114304  28.574404  146.474625  63.928844  72.699326
2019-11-25 00:00:00  78.225403  28.883261  147.662201  65.049736  74.144035
2019-11-26 00:00:00  78.897903  28.943041  145.750885  64.541801  74.110245
2019-11-27 00:00:00  79.024597  28.843410  147.318893  65.408730  74.625610
2019-11-29 00:00:00  78.722458  28.634182  145.741608  65.264648  74.118683

                             ABC        ABT        ACGL         ACN        ADBE  \
Date
2019-11-22 00:00:00  85.750618  78.712914  40.840000  186.436661  299.299988
2019-11-25 00:00:00  86.111610  79.417862  41.200001  188.048294  305.279999
2019-11-26 00:00:00  83.460999  80.292030  42.080002  190.475235  307.899994
2019-11-27 00:00:00  84.088028  80.292030  42.230000  190.788071  309.059998
2019-11-29 00:00:00  83.518013  80.320244  41.970001  190.702789  309.529999

                     …        WYNN        XEL        XOM       XRAY  \
Date                 …
2019-11-22 00:00:00  …  117.900528  55.215652  57.267715  55.027195
2019-11-25 00:00:00  …  122.659691  54.801044  56.887985  55.401920
2019-11-26 00:00:00  …  120.265266  55.143555  56.747612  54.806194
2019-11-27 00:00:00  …  120.680824  55.486057  56.714607  54.873463
```

```
2019-11-29 00:00:00  …  119.572662  55.422970  56.244041  54.325790

                            XYL         YUM          ZBH         ZBRA        ZION  \
Date
2019-11-22 00:00:00  73.728554  91.570831  135.793152  243.570007  44.547134
2019-11-25 00:00:00  73.440880  91.936661  136.982819  248.940002  44.734386
2019-11-26 00:00:00  73.977867  93.568848  136.775116  247.699997  44.279640
2019-11-27 00:00:00  74.418961  94.403694  138.380203  255.210007  44.645222
2019-11-29 00:00:00  74.323059  94.431839  137.171631  250.940002  44.386642

                             ZTS
Date
2019-11-22 00:00:00  116.827705
2019-11-25 00:00:00  117.607796
2019-11-26 00:00:00  118.855934
2019-11-27 00:00:00  118.602402
2019-11-29 00:00:00  117.520035

[5 rows x 496 columns]
[**********************100%**********************]  1 of 1 completed
```

7 Function 1

This following function calculates what return we will recieve based on our weightings. The function carefully considers that the weightings have been decided based on previous information up to and including this current day, which is how the optimal weightings have been calculated, as this is all the infomation we know. Then these weightings will be applied to the stock for the next day, where the returns are currently unknown, until tomorrow occurs. This is the trading logic behind this model once the weightings have been updated.

From this, the daily returns are stored plotted later in the notebook, the portfolio balance is also updated through this function.

Each individual equitiy that we have considered, returns' are calculated under the returnsX variable (row vector, each column in this row vector represents an individual equity). Here the value at the next point in time is subtracted from the value at the current point in time, and then divided by the current point in times value (view equation 1 below). Expressing the change in value as a ratio, rather then a percentage, which comes later. Then each change in value is multiplied by its corresponding weight which has been determined by maximising the sharpe ratio. e.g. 50% weight in one equity which increases by 1% and 50% weight in another equity which also increases by 1%, means: Return=0.5x1+0.5x1=1% overall return. So when each return has been multipled by its' corresponding weight, we then sum these results via the np.dot operator (view equation 2 below). The balance is then updated, as the balance will have been invested amoungst the various equities and therefore needs to be updated appropraitely (equation 3 below) via Balance=Balance+Balance*(1+Total_ret) (at this point Total_ret is still not a percentage, but a ratio). E.g. Balance=£1,000 and our Total_ret for the current day which has just passed is -0.05, therefore we will lose £50. So the new_balance = old_balance - £50.

After the balance has been updated, Total_ret is then turned from a ratio to a percentage.

Additionally, if short selling is enabled, then the weightings for certain securities (equities in this case) are allowed to be negative. Therefore you may question if this function is still feasible, even with negative weightings. The answer is yes, as when a weighting is negative (predicitng a downtrend in price action), and the price does decrease, the returnsX element for that asset will now be negative. When multiplying the 'weights' and 'returnsX', a -ve (from the weight) multiplied with a -ve (from the drop in asset price) results in a +ve value, meaning the short sale was successfully considered in this function, this +ve value then proceeds to form a sum called the 'Total_ret', and the function continues as discussed earlier.

Equation 1 Where, PA_{t+1} is the price action at the next point in time (next day represented by the next row in our dataframe). PA_t is the price action at the current point in time.

$$Return = \frac{PA_{t+1} - PA_t}{PA_t}$$

Equation 2 Where w_i represents an individual equity assigned weight. r_i represents an individual equities return for the given period of investment (single day). $\mathbf{W}$ represents a column vector of the weights for each equity. $\mathbf{R}$ represents a row vector of the returns for each equity. n represents the number of equities our portfolio considers.

$$(\mathbf{W} \cdot \mathbf{R})_t = \sum_{i=1}^{n} w_i r_i$$

Equation 3 Bal_{t+1} is the balance of the portfolio at the next point in time, based on our returns. Bal_t is the balance of the portfolio at the current point in time, which is about to be updated based on our investing results for the upcoming day. $(\mathbf{W} \cdot \mathbf{R})_t$ is the returns based on our returns and weights for each equity.

$$Bal_{t+1} = Bal_t(1 + (\mathbf{W} \cdot \mathbf{R})_t)$$

[20]:
```
def Daily_return_cal(df, weights, i, N, Balance, Win_trades, Lose_trades):
    returnsX=((df.iloc[N+i,:]
               -df.iloc[N+i-1,:])
               /df.iloc[N+i-1,:])
    Total_ret=0

    Total_ret = np.dot(weights, returnsX)

    Balance+=Balance*(Total_ret)

    Total_ret=Total_ret*100

    if Total_ret>0:
        Win_trades+=1
    else:
        Lose_trades+=1

    return returnsX, Total_ret, Balance, Win_trades, Lose_trades
```

8 Function 2

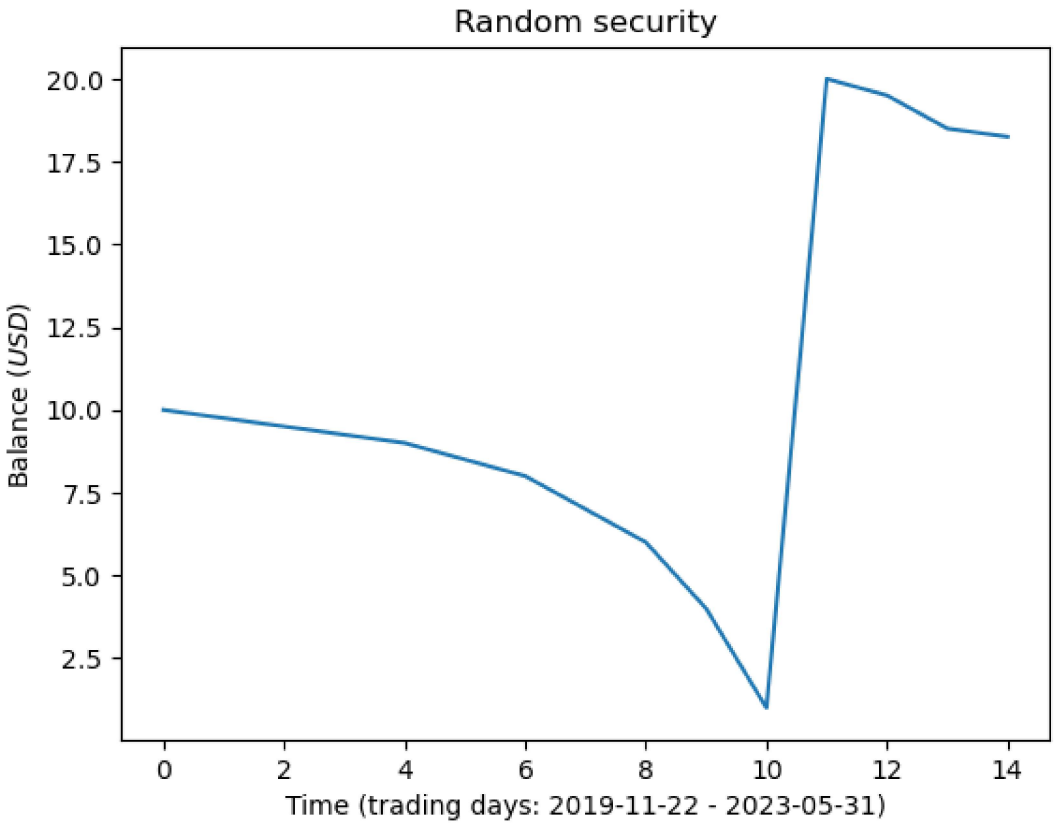
Orginally when attempting to run this code. There was an issue related to solving the covariance matrix, where the matrix would often result in being a non-positive semi definite, meaning one of the eigenvalues is zero or negative, leading to issues with the solving mean-variance optimisation (MVO) problem. As desrcibed in the above sections; further MPT explanation and MPT issues, we need to take an inverse of the covariance matrix, this is not suitable if we have a covariance matrix which is non-positive semi definite or non-positive definite.

Therefore I will apply the Ledoit-Wolf shrinkage in an attempt to resolve this issue. This method applies shrinkage to the sample covariance matrix, reducing the impact of noise and improving stability. It strikes a balance between the sample covariance and a known target covariance matrix, usually the identity matrix. This technique is beneficial when dealing with limited historical data or a large number of assets. We have a limited number of observations, with a large number of dimensions, in this case each dimension is represented by an individual equity from the S&P 500.

[21]:
```
def Ledoit_Wolf(df, N,i):
    lw_shrinkage = LedoitWolf()

    lw_shrinkage.fit(
        (df.iloc[i:N+i,:])
        .to_numpy()
        )
    S_LW = lw_shrinkage.covariance_
    return S_LW
```

9 Function 3

[52]:
```
y=[10,9.75,9.5,9.25,9,8.5,8,7,6,4,1,20,19.5,18.5,18.25]
plt.plot(y)
plt.xlabel('Time (trading days: 2019-11-22 - 2023-05-31)')
plt.ylabel('Balance ($USD$)')
plt.title('Random security')
plt.show()
```



The above graph is simply to represent the considerations when calculating the maximum equity drawdown. Firstly, the lowest point in our portfolio value, must have a value greater then itself, at a point prior in time, hence the term 'lookbardward' used in the function below. Secondly, after our portfolio reaches a maximum value, the securitiy could drop significantly, or not significantly, either way this needs to be checked for by 'lookfoward', in coding syntax, 'max_index_lookforward+1:'. Then the maximum between these two scenarios will be taken as the maxiumum equity drawdown of our portfolio.

This function is only for post-processing the results. Here we calculated the maximum equity drawdown which our portfolio encounters. The maximum equity drawdown is the worst position our portfolio capital could experience.

[23]:
```
def Max_equity_drawdown_calc(Bal_vs_time_Part1):
    max_val_lookforward=max(Bal_vs_time_Part1)
    max_index_lookforward=Bal_vs_time_Part1.index(max_val_lookforward)
    min_val_lookforward=min(Bal_vs_time_Part1[max_index_lookforward+1:])

    min_val_lookbackward=min(Bal_vs_time_Part1)
    min_index_lookbackward=Bal_vs_time_Part1.index(min_val_lookbackward)
    max_val_lookbackward=max(Bal_vs_time_Part1[:min_index_lookbackward-1])
```

```
    lookforward_equity_drawdown=(max_val_lookforward-min_val_lookforward)/
↪max_val_lookforward
    lookbackward_equity_drawdown=(max_val_lookbackward-min_val_lookbackward)/
↪max_val_lookbackward

    equity_drawdown=round(max(lookforward_equity_drawdown,␣
↪lookbackward_equity_drawdown)*100,1)
    return equity_drawdown
```

10 Calculating returns

The returns are calculated and the first row with value NA is dropped, the df first row is also dropped, so the indexing refers to the same dates, avoiding look-ahead bias. This is import for calculating the covariance matrix within the main loop at a later stage.

[24]:
```
returns=df.pct_change()
returns = returns.drop(returns.index[0])
df = df.drop(df.index[0])
returns.replace(0, 1e-6, inplace=True)
returns_log = np.log(1+returns)
returns_log.replace(0, 1e-6, inplace=True)
```

11 Defining some variables

For these variables, there is a Part 1, Part 2, Part 3 and Part 4 which was discussed in the opening comments. Part 1 and 2 variables are associated with a regular MPT which is subjected to the mean-variance tradeoff issue. Part 3 variables are associated with an optimised MPT with L2 regularisation incorporated. Then Part 1 calculates the covariance via the sample covariance, whilst Parts 2 and 3 use the Ledoit-Wolf shrinkage regularisation for solving the covariance matrix.

Here Balance represents a demo account with starting capital of $10,000. Bal_vs_time is a list which is to be appended to. Each trading day, the newly updated balance is stored into Bal_vs_time, which can then be plotted after the main loop has finished. Returns is a list which is appended to, and stores the daily returns our portfolio makes as a percentage, which is plotted later in the script.

There are two lookback periods. N_returns, and N_covariance. N_returns works with calculating the mean return over a given lookback period, having a high N_returns can result in a serverely lagging dataset. In contrast, our lookback period for calculating the covaraince matrix needs sufficient observations, therefore N_covariance is significantly greater than N_returns.

Win_trades and Lose_trades are counters, so we can analyse how made trades are successful (generate a return greater than 0 percent).

[25]:
```
N_returns=30
N_covariance=400

Bal_vs_time_Part1=[]
```

```
Balance_Part1=10000
Returns_Part1=[]
Lose_trades_Part1=0
Win_trades_Part1=0

Bal_vs_time_Part2=[]
Balance_Part2=10000
Returns_Part2=[]
Lose_trades_Part2=0
Win_trades_Part2=0

Bal_vs_time_Part3=[]
Balance_Part3=10000
Returns_Part3=[]
Lose_trades_Part3=0
Win_trades_Part3=0

Bal_vs_time_Part4=[]
Balance_Part4=10000
Returns_Part4=[]
Lose_trades_Part4=0
Win_trades_Part4=0
returns_spy = spy_data.pct_change()
```

12 Main loop

This loop runs through all the historical data, feeding the input data into the model day by day, trying to replicate a real time scenario. Therefore this loop iterates through the rows, where each row represents a single day. Here, the key variables calculated are covariance and expected returns. The weights are determined by finding the portfolio which can achieve the greatest sharpe ratio. Then Function 1 is called once the weights have been found for the current point in time. This function as discussed earlier, will then use these weightings and invest our account balance into the specific equities which are believed to achieve good results. Within this function, it calculates for us the returns we will get for the next day, and produce the key variables: 'Total_ret' and 'Balance' which can be appended to lists, and plotted later for a visual representation of how this model is preforming.

Note: the risk free rate is not a representation of the current risk-free rate. This is not required, as the back end script simply holds a risk free rate constant at the same value when testing the various portfolios, to make it a fair test. Whilst the calcuated sharpe ratio may not be accurate, it is fair and relative to all the other various portfolio weightings as it is held at a constant value. So the sharpes ratios are relative to all the alternative weighted portfolios. Here the risk free rate in the back end code is considered at 0.02 (2% per annum), this **pypfopt** package has built an MPT for an annual rebalance period. Therefore with 252 trading days each year (US), the risk free rate which has to be exceeded to weights to be calculated is now =0.02/252.

Note: Certain packages with this MPT will run into errors when solving for the sharpe ratio. These packages work assuming that the expected returns will always be greater than the risk free rate.

When this is not the case, this package believes the maximum sharpe ratio can not be sovled for and an error occurs, breaking the code. Therefore the following additional script has been added, allowing the code to continue to the next day if an error has occured, or the maximum sharpe ratio 'can not' be found.

except Exception as e: print(f"Error occurred at index {i}: {str(e)}") continue

This allows the code to pass to the next day and continue as normal. On days where the sharpe ratio can not be solved for, the function: 'Daily_return_calc' is not called, so our portfolio will miss days of investing due to no optimal distrubution of weightings being found. The above code also prints a statement highlighting the specific days where a maximum sharpe ratio can not be solved for. This occurs when the mean expected returns for all of the equities is below then the risk free rate (this is unlikely to occur with so many equities considered).

[26]:
```
for i in range(0, len(df)-N_covariance):
    try:

        mu = (returns.iloc[N_covariance+i-N_returns:N_covariance+i,:]).mean()
        S = (returns_log.iloc[i:N_covariance+i,:]).cov()
        S_LW=Ledoit_Wolf(returns_log, N_covariance, i)

        ##### Part 1 - MPT with mean-variance issue present - pypfopt#####

        ef_Part1 = EfficientFrontier(mu, S, solver=SCS)
        weights_calc = ef_Part1.max_sharpe(risk_free_rate=0.02/252)
        weights1 = np.array([value for value in weights_calc.values()])

        (returnsX_Part1,
        Total_ret_Part1,
        Balance_Part1,
        Win_trades_Part1,
        Lose_trades_Part1)=Daily_return_cal(
                                    df,
                                    weights1,
                                    i,
                                    N_covariance,
                                    Balance_Part1,
                                    Win_trades_Part1,
                                    Lose_trades_Part1
                                    )
        Returns_Part1.append(Total_ret_Part1)
        Bal_vs_time_Part1.append(Balance_Part1)

    except Exception as e:
        print(f"Error occurred at index {i}: {str(e)}")


        #### Part 2- MPT with mean vairance issue present and
```

```
        #    Ledoit-Wolf cov shrinkage ####
    try:
        ef_Part2 = EfficientFrontier(mu, S_LW, solver=ECOS)
        weights_calc = ef_Part2.max_sharpe(risk_free_rate=0.02/252)
        weights2 = np.array([value for value in weights_calc.values()])

        (returnsX_Part2,
        Total_ret_Part2,
        Balance_Part2,
        Win_trades_Part2,
        Lose_trades_Part2)=Daily_return_cal(
                                    df,
                                    weights2,
                                    i,
                                    N_covariance,
                                    Balance_Part2,
                                    Win_trades_Part2,
                                    Lose_trades_Part2
                                    )


        Returns_Part2.append(Total_ret_Part2)
        Bal_vs_time_Part2.append(Balance_Part2)

    except Exception as e:
        print(f"Error occurred at index {i}: {str(e)}")

        #### Part 3 - MPT attempt MVO issue removed - Ledoit wolf shrinkage
        #    for solving the cov matrix ####
    try:
        ef_Part3 = EfficientFrontier(mu, S_LW, solver=SCS)
        ef_Part3.add_objective(objective_functions.L2_reg, gamma=2)
        weights_calc = ef_Part3.max_sharpe(risk_free_rate=0.02/252)
        weights3 = np.array([value for value in weights_calc.values()])
        weights3 = np.clip(weights3, a_min=0, a_max=None)

        (returnsX_Part3,
        Total_ret_Part3,
        Balance_Part3,
        Win_trades_Part3,
        Lose_trades_Part3)=Daily_return_cal(
                                    df,
                                    weights3,
                                    i,
                                    N_covariance,
                                    Balance_Part3,
                                    Win_trades_Part3,
```

```
                                    Lose_trades_Part3
                                    )


        Returns_Part3.append(Total_ret_Part3)
        Bal_vs_time_Part3.append(Balance_Part3)

    except Exception as e:
        print(f"Error occurred at index {i}: {str(e)}")


        #### SPY investment ###
    try:
        Balance_Part4 =Balance_Part4*(1+returns_spy.iloc[i+N_covariance-1])
        Bal_vs_time_Part4.append(Balance_Part4)

        if returns_spy.iloc[i+N_covariance-1]>0:
            Win_trades_Part4+=1

        elif returns_spy.iloc[i+N_covariance-1]<0:
            Lose_trades_Part4+=1


    except Exception as e:
        print(f"Error occurred at index {i}: {str(e)}")
        continue
```

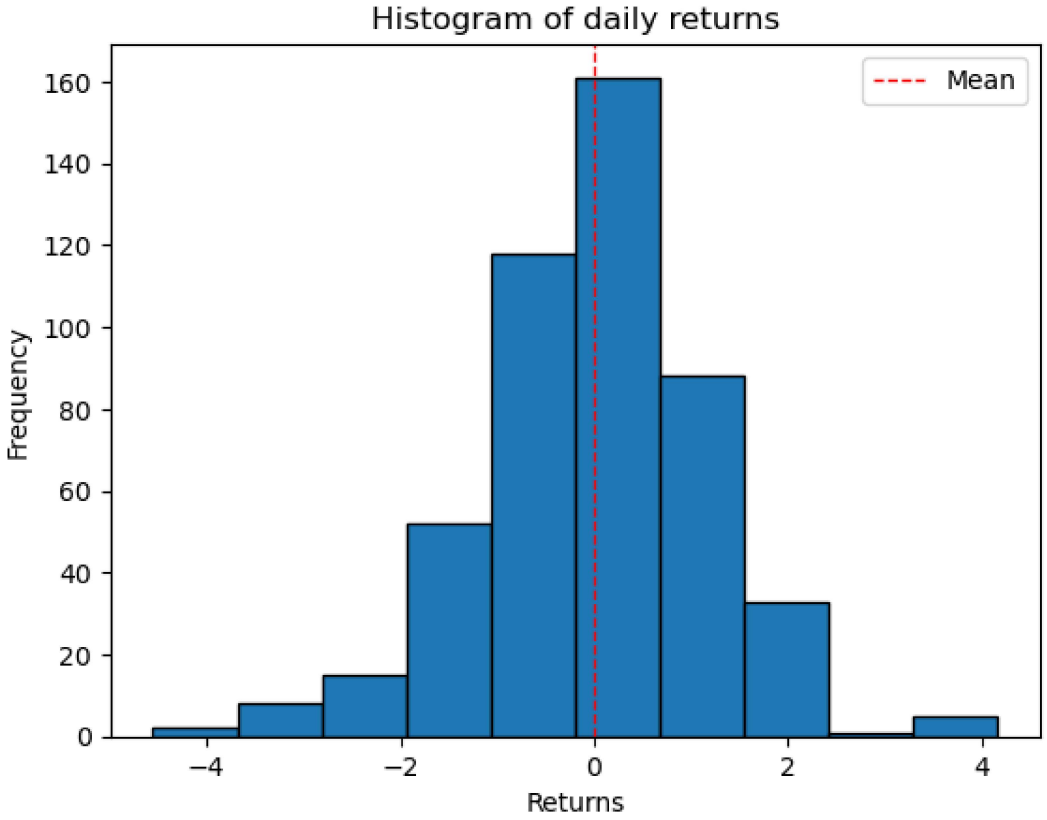
```
Error occurred at index 0: ARPACK error -1: ARPACK error -1: No convergence
(4961 iterations, 0/1 eigenvectors converged)


        CVXPY note: This failure was encountered while trying to certify
        that a matrix is positive semi-definite (see [1] for a definition).
        In rare cases, this method fails for numerical reasons even when the
matrix is
        positive semi-definite. If you know that you're in that situation, you
can
        replace the matrix A by cvxpy.psd_wrap(A).

        [1] https://en.wikipedia.org/wiki/Definite_matrix

Error occurred at index 1: ARPACK error -1: ARPACK error -1: No convergence
(4961 iterations, 0/1 eigenvectors converged)


        CVXPY note: This failure was encountered while trying to certify
        that a matrix is positive semi-definite (see [1] for a definition).
        In rare cases, this method fails for numerical reasons even when the
```

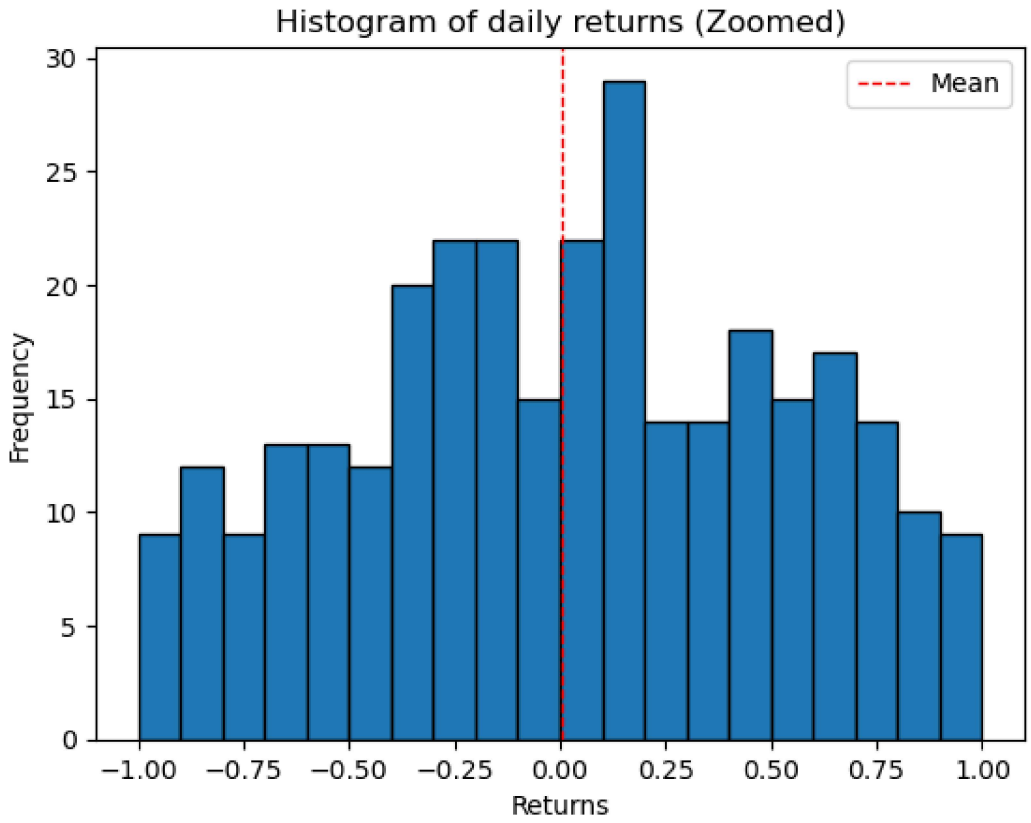
```
        CVXPY note: This failure was encountered while trying to certify
        that a matrix is positive semi-definite (see [1] for a definition).
        In rare cases, this method fails for numerical reasons even when the
matrix is
        positive semi-definite. If you know that you're in that situation, you
can
        replace the matrix A by cvxpy.psd_wrap(A).

        [1] https://en.wikipedia.org/wiki/Definite_matrix
```

13 Post-processing results - Part 1

Graphical histogram of the potential returns we could expect to make on a daily time frame.

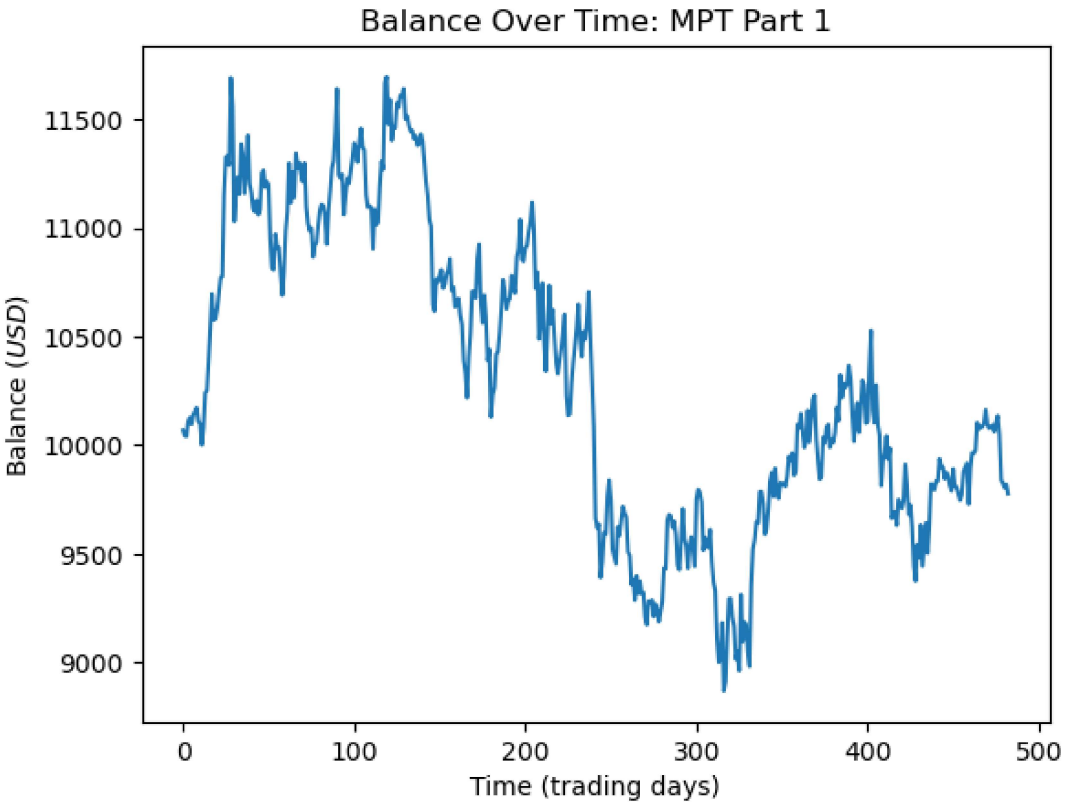
[32]:
```
### Plot 1 ###
plt.hist(Returns_Part1, bins=10, edgecolor='black')
plt.xlabel('Returns')
plt.ylabel('Frequency')
plt.title('Histogram of daily returns')

mean = sum(Returns_Part1) / len(Returns_Part1)
plt.axvline(mean, color='red', linestyle='dashed', linewidth=1, label='Mean')
plt.legend()
plt.show()

### Plot 2 ###
plt.hist(Returns_Part1, bins=20, edgecolor='black', range=(-1.0, 1.0))
plt.xlabel('Returns')
plt.ylabel('Frequency')
plt.title('Histogram of daily returns (Zoomed)')

mean = sum(Returns_Part1) / len(Returns_Part1)
plt.axvline(mean, color='red', linestyle='dashed', linewidth=1, label='Mean')
plt.legend()

# Show the plot
plt.show()
```

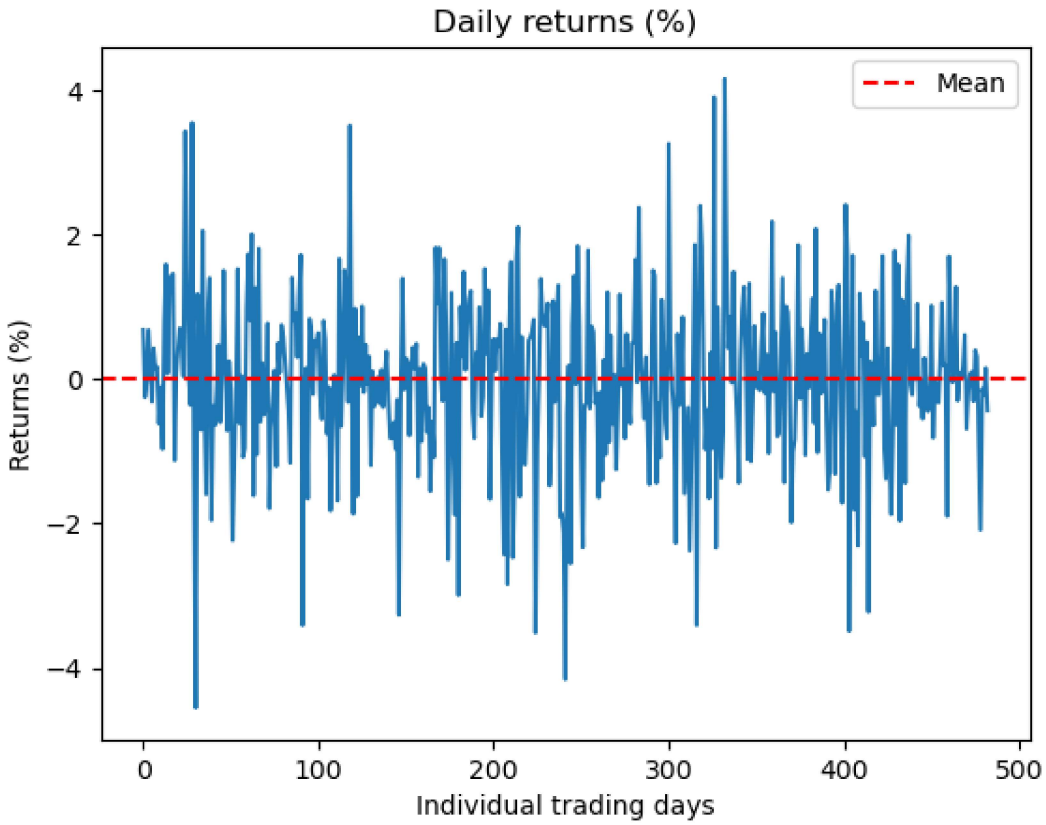

Histogram of daily returns
Mean
Frequency
Returns
0
20
40
60
80
100
120
140
160
−4
−2
0
2
4



Portfolio balance vs trading days. Regular MPT.

[53]:
```
plt.plot(Bal_vs_time_Part1)
plt.xlabel('Time (trading days)')
plt.ylabel('Balance ($USD$)')
plt.title('Balance Over Time: MPT Part 1')
plt.show()

plt.plot(Returns_Part1)
plt.xlabel('Individual trading days')
plt.ylabel('Returns (%)')
plt.title('Daily returns (%)')
mean_returns = sum(Returns_Part1) / len(Returns_Part1)
plt.axhline(mean_returns, color='red', linestyle='--', label='Mean')
plt.legend()
plt.show()
```


Balance Over Time: MPT Part 1
Balance (USD)
11500
11000
10500
10000
9500
9000
0
100
200
300
400
500
Time (trading days)



Winning trades meaning the return for that day was positive. And losing trades meaning the return for that day was negative. The maximum equity drawdown for the whole portfolio is considered and the maximum equity drawdown for our single biggest losing trade is considered.

[34]:
```
print('Number of winning trades: ', Win_trades_Part1)
print('Number of losing trades: ', Lose_trades_Part1)
print('Total number of trades made: ', (Win_trades_Part1+Lose_trades_Part1))
equity_drawdown=Max_equity_drawdown_calc(Bal_vs_time_Part1)
print('Maximum equity drawdown for the whole portfolio: ', equity_drawdown, '␣
↪%')
print('Maximum equity drawback from a single trade: ',
     round(min(Returns_Part1),1), 'as a percentage of equity invested.')
```

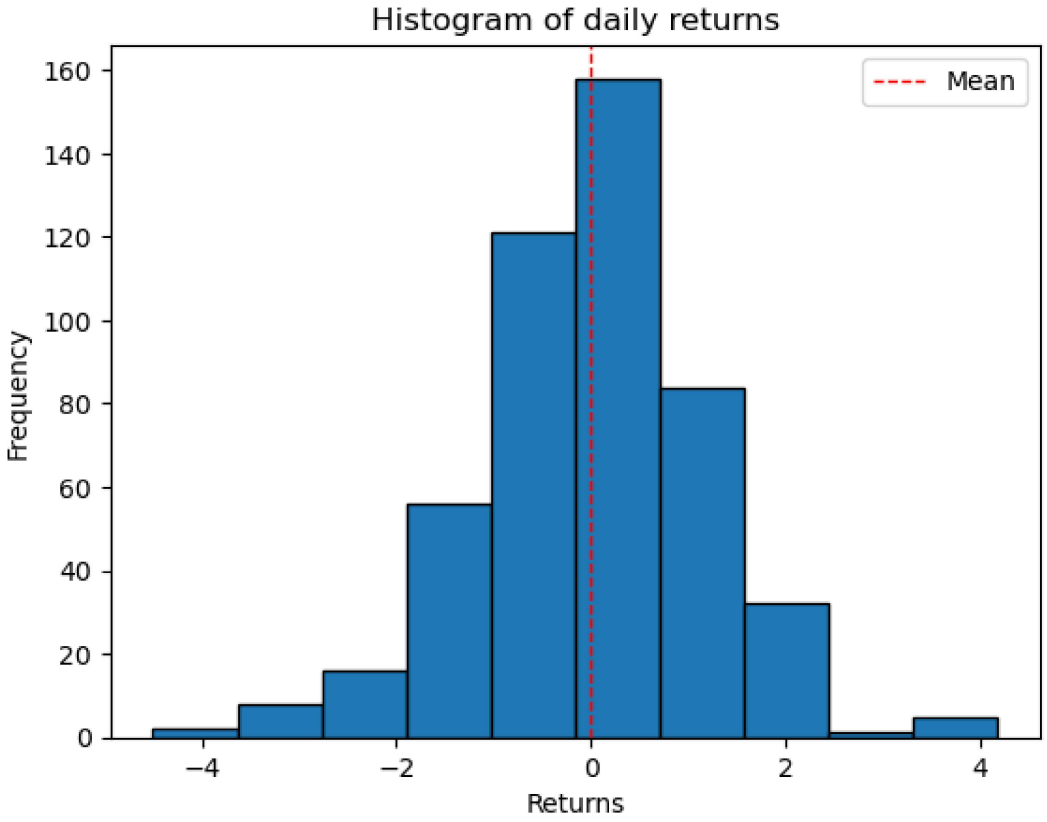
```
Number of winning trades:  252
Number of losing trades:  231
Total number of trades made:  483
Maximum equity drawdown for the whole portfolio:  24.2  %
Maximum equity drawback from a single trade:  -4.6 as a percentage of equity
invested.
```

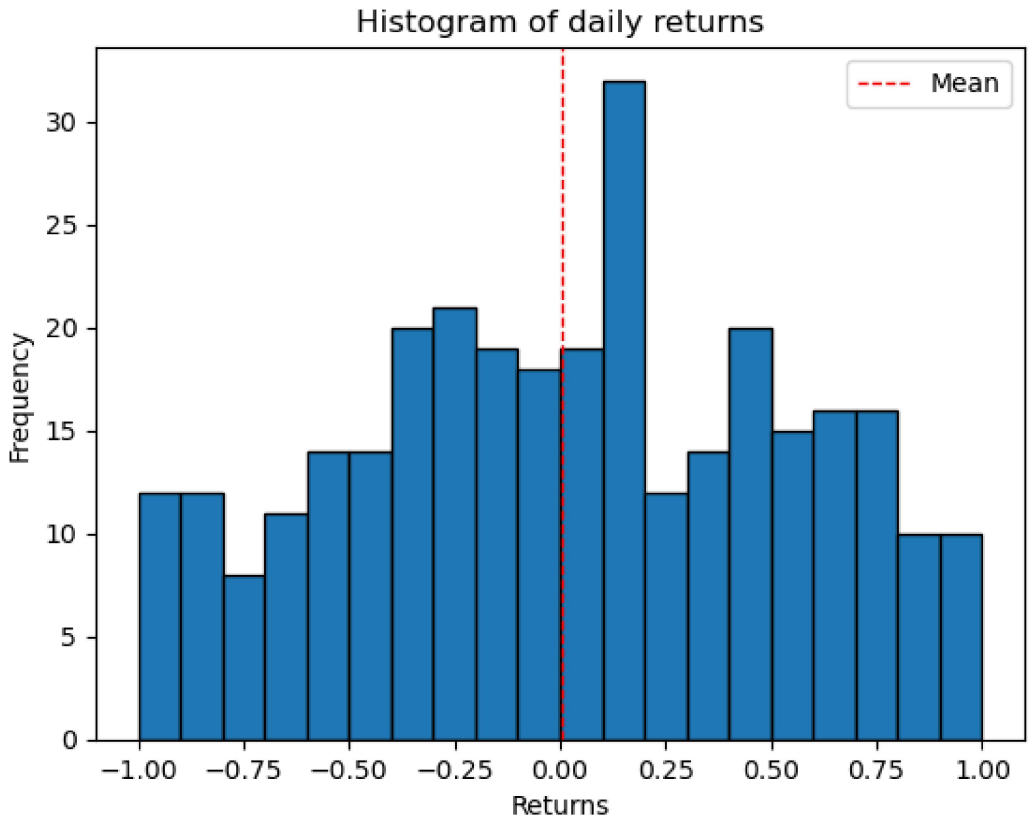
14 Post-processing results - Part 2

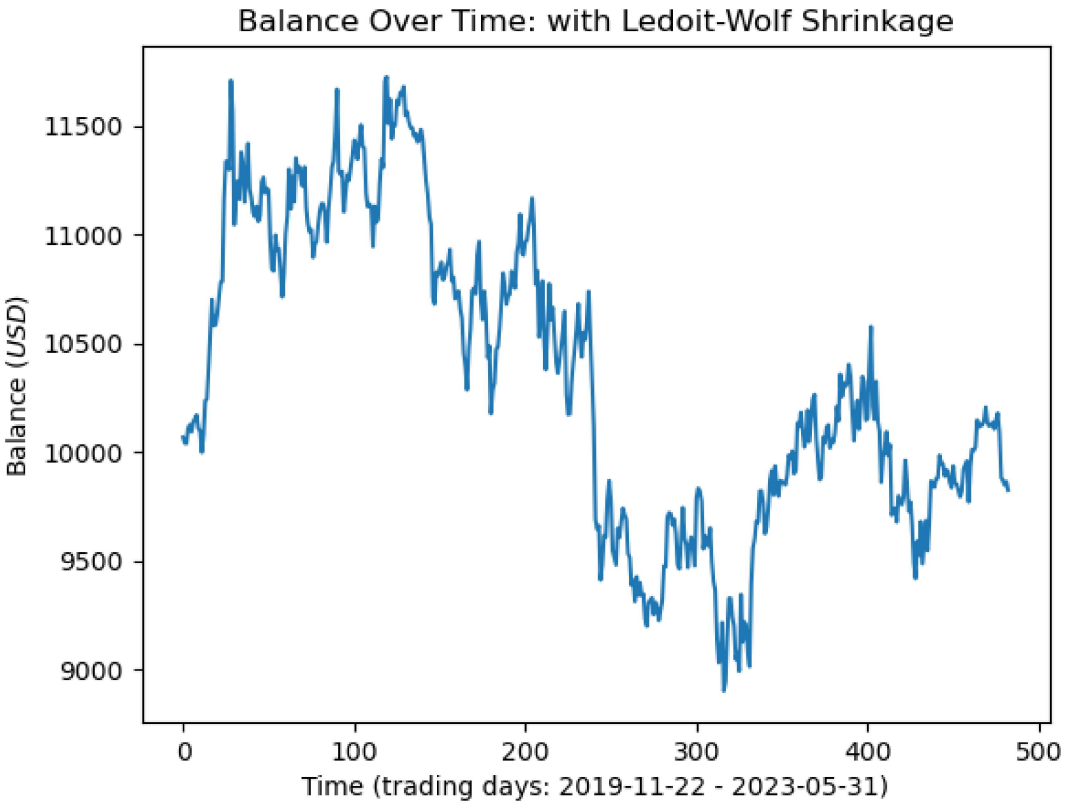
Here I will repeat the code above in a single cell. But I will summarise the results in the Comparison stage.

[50]:
```
### Plot 1 ###
plt.hist(Returns_Part2, bins=10, edgecolor='black')
plt.xlabel('Returns')
plt.ylabel('Frequency')
plt.title('Histogram of daily returns')
mean = sum(Returns_Part2) / len(Returns_Part2)
plt.axvline(mean, color='red', linestyle='dashed', linewidth=1, label='Mean')
plt.legend()
plt.show()


### Plot 2 ###
plt.hist(Returns_Part2, bins=20, edgecolor='black', range=(-1.0, 1.0))
plt.xlabel('Returns')
plt.ylabel('Frequency')
plt.title('Histogram of daily returns')
mean = sum(Returns_Part2) / len(Returns_Part2)
plt.axvline(mean, color='red', linestyle='dashed', linewidth=1, label='Mean')
plt.legend()
plt.show()

plt.plot(Bal_vs_time_Part2)
plt.xlabel('Time (trading days: 2019-11-22 - 2023-05-31)')
plt.ylabel('Balance ($USD$)')
plt.title('Balance Over Time: with Ledoit-Wolf Shrinkage')
plt.show()

print('Number of winning trades: ', Win_trades_Part2)
print('Number of losing trades: ', Lose_trades_Part2)
print('Total number of trades made: ', (Win_trades_Part2+Lose_trades_Part2))
equity_drawdown=Max_equity_drawdown_calc(Bal_vs_time_Part2)
print('Maximum equity drawdown for the whole portfolio: ',equity_drawdown,
      ' %')
print('Maximum equity drawback from a single trade: ',
     round(min(Returns_Part2),1), 'as a percentage of equity invested.')

plt.plot(Returns_Part2)
plt.xlabel('Individual trading days')
plt.ylabel('Returns (%)')
plt.title('Daily returns (%)')
mean_returns = sum(Returns_Part2) / len(Returns_Part2)
plt.axhline(mean_returns, color='red', linestyle='--', label='Mean')
plt.legend()
```

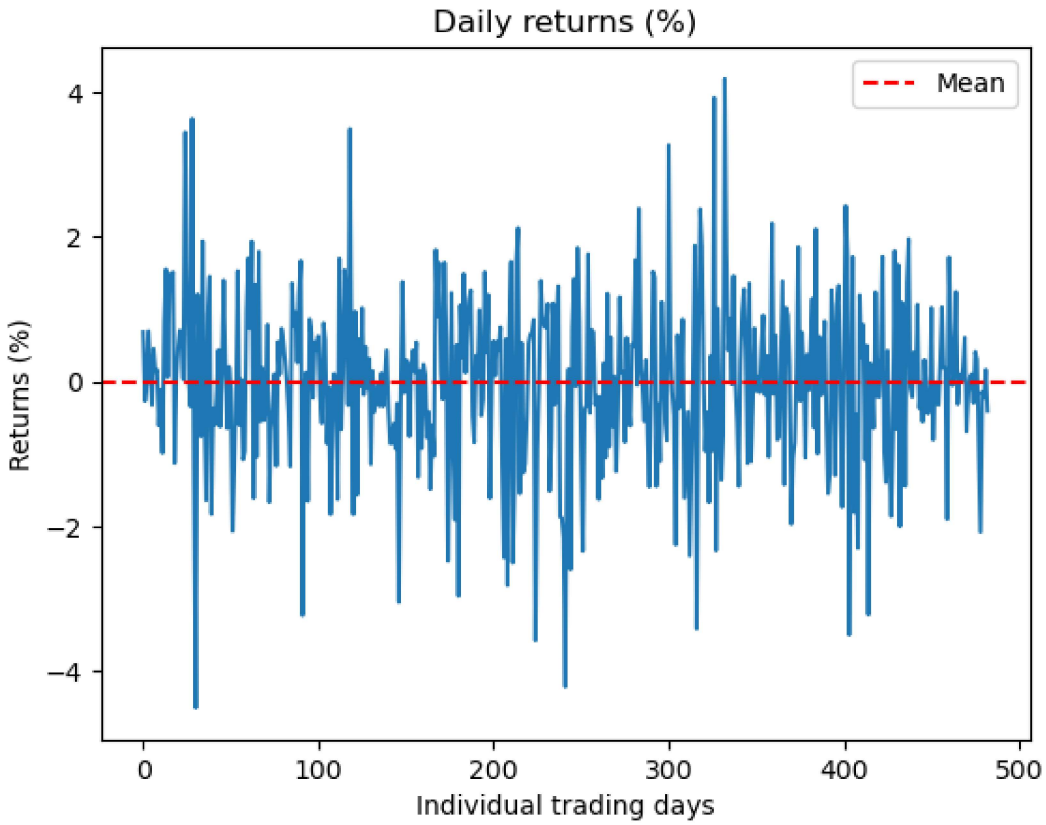
```
plt.show()
```


Histogram of daily returns
Mean
Frequency
Returns
160
140
120
100
80
60
40
20
0
−4
−2
0
2
4


Histogram of daily returns
Mean
Frequency
Returns
0
5
10
15
20
25
30
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00




```
Number of winning trades:  251
Number of losing trades:  232
Total number of trades made:  483
Maximum equity drawdown for the whole portfolio:  24.1  %
Maximum equity drawback from a single trade:  -4.5 as a percentage of equity
invested.
```

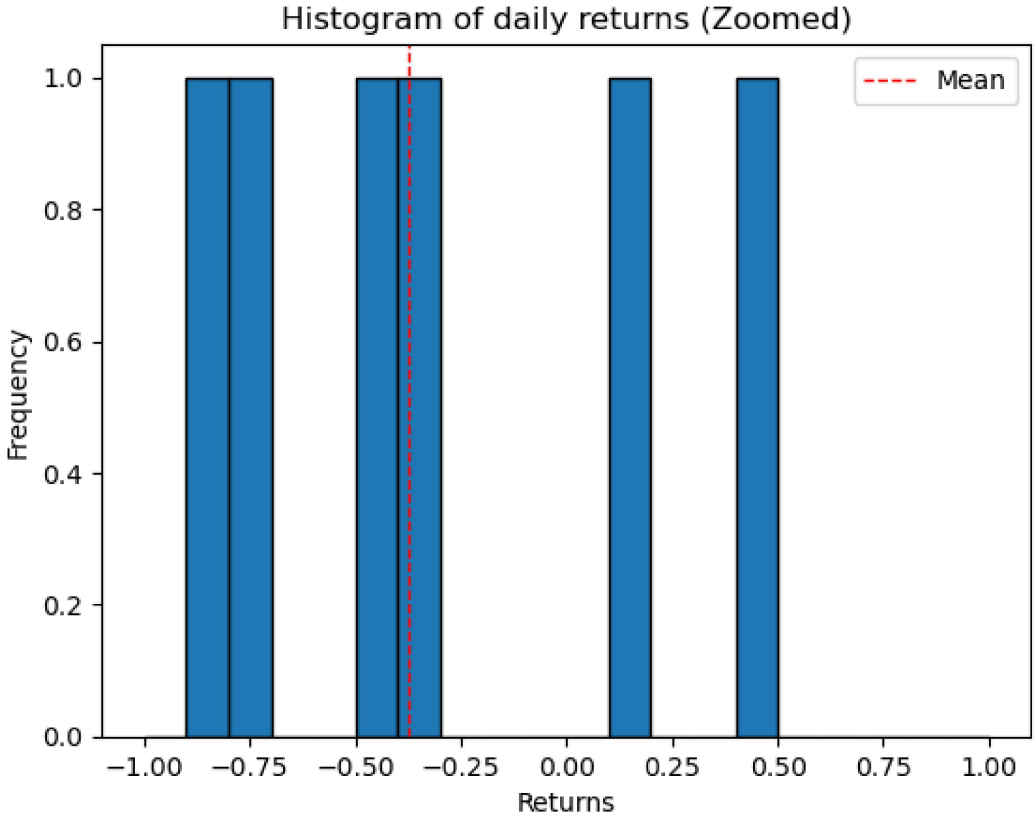


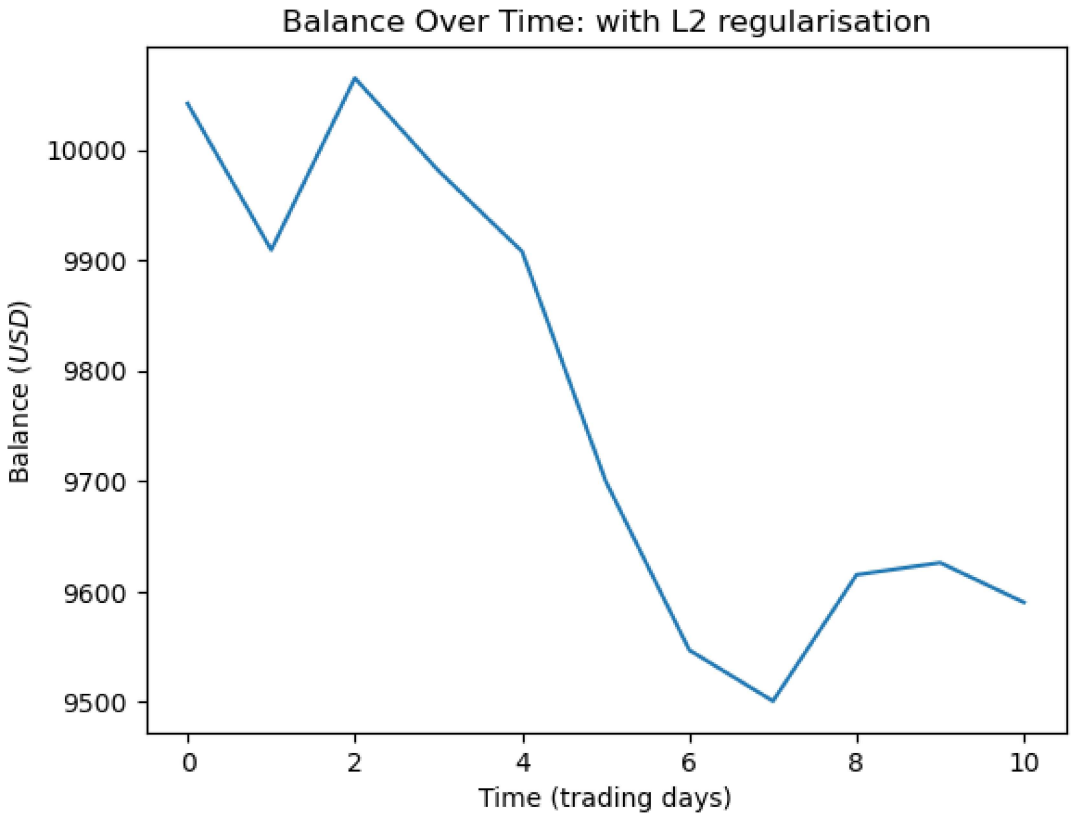
15 Post-processing results - Part 3

[51]:
```
### Plot 1 ###
plt.hist(Returns_Part3, bins=10, edgecolor='black')
plt.xlabel('Returns')
plt.ylabel('Frequency')
plt.title('Histogram of daily returns')


mean = sum(Returns_Part3) / len(Returns_Part3)
plt.axvline(mean, color='red', linestyle='dashed', linewidth=1, label='Mean')
plt.legend()

plt.show()


### Plot 2 ###
plt.hist(Returns_Part3, bins=20, edgecolor='black', range=(-1.0, 1.0))
plt.xlabel('Returns')
```

```
plt.ylabel('Frequency')
plt.title('Histogram of daily returns (Zoomed)')

mean = sum(Returns_Part3) / len(Returns_Part3)
plt.axvline(mean, color='red', linestyle='dashed', linewidth=1, label='Mean')
plt.legend()

# Show the plot
plt.show()

plt.plot(Bal_vs_time_Part3)
plt.xlabel('Time (trading days)')
plt.ylabel('Balance ($USD$)')
plt.title('Balance Over Time: with L2 regularisation')
plt.show()

print('Number of winning trades: ', Win_trades_Part3)
print('Number of losing trades: ', Lose_trades_Part3)
print('Total number of trades made: ', (Win_trades_Part3+Lose_trades_Part3))
equity_drawdown=Max_equity_drawdown_calc(Bal_vs_time_Part3)
print('Maximum equity drawdown for the whole portfolio: ', equity_drawdown,
      ' %')
print('Maximum equity drawback from a single trade: ',
     round(min(Returns_Part3),1), 'as a percentage of equity invested.')
```


Histogram of daily returns
Mean
Frequency
Returns
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Histogram of daily returns (Zoomed)
Mean
Frequency
Returns
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

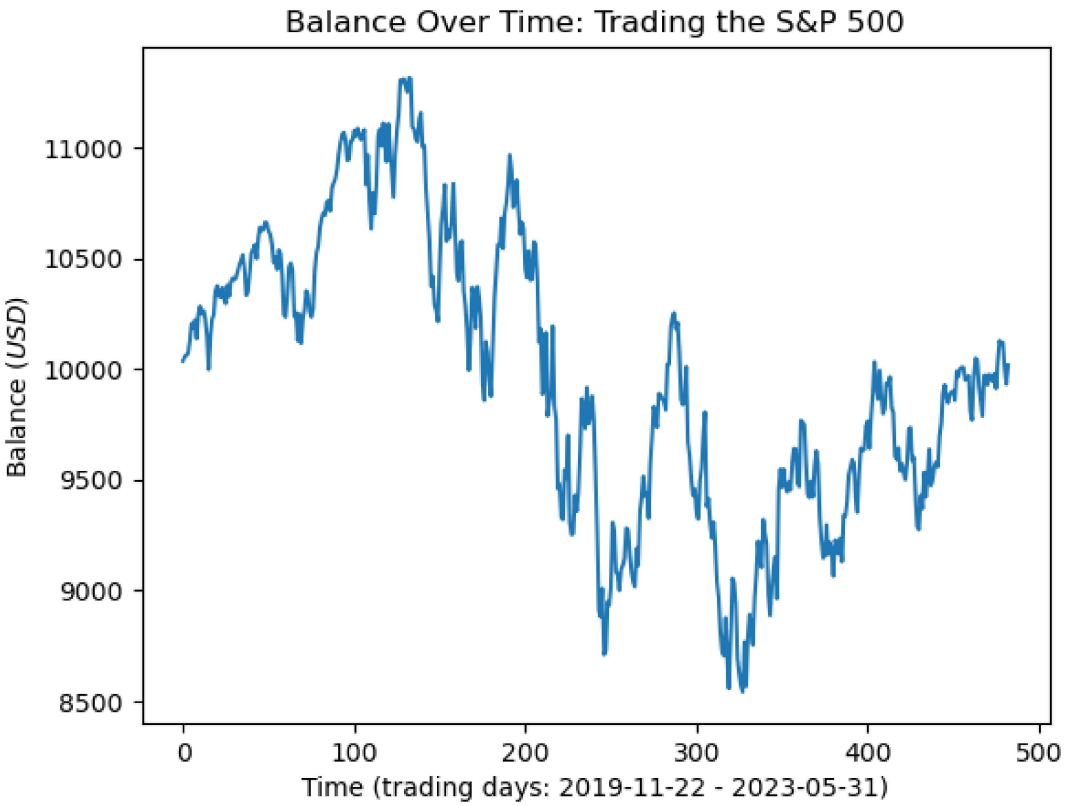



```
Number of winning trades:  4
Number of losing trades:  7
Total number of trades made:  11
Maximum equity drawdown for the whole portfolio:  5.6  %
Maximum equity drawback from a single trade:  -2.1 as a percentage of equity
invested.
```

16 Post-processing results - Part 4

[58]:
```
plt.plot(Bal_vs_time_Part4)
plt.xlabel('Time (trading days: 2019-11-22 - 2023-05-31)')
plt.ylabel('Balance ($USD$)')
plt.title('Balance Over Time: Trading the S&P 500')
plt.show()

print('Number of winning trades: ', Win_trades_Part4)
print('Number of losing trades: ', Lose_trades_Part4)
print('Total number of trades made: ', (Win_trades_Part4+Lose_trades_Part4))
equity_drawdown=Max_equity_drawdown_calc(Bal_vs_time_Part4)
print('Maximum equity drawdown for the whole portfolio: ', equity_drawdown,
```

```
      ' %')
```

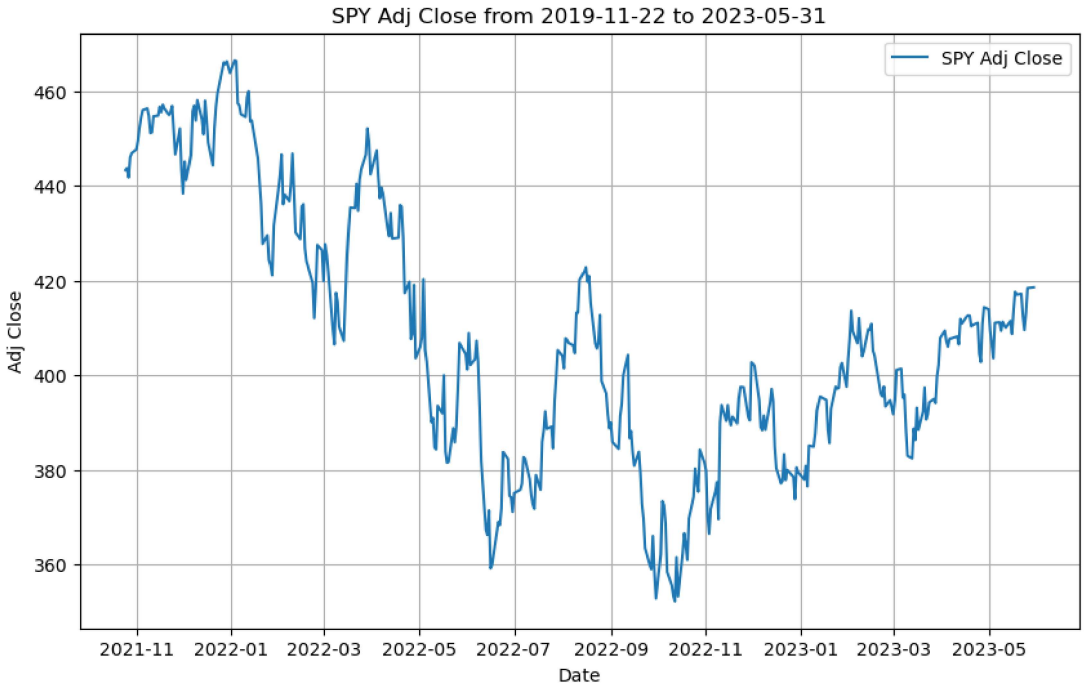


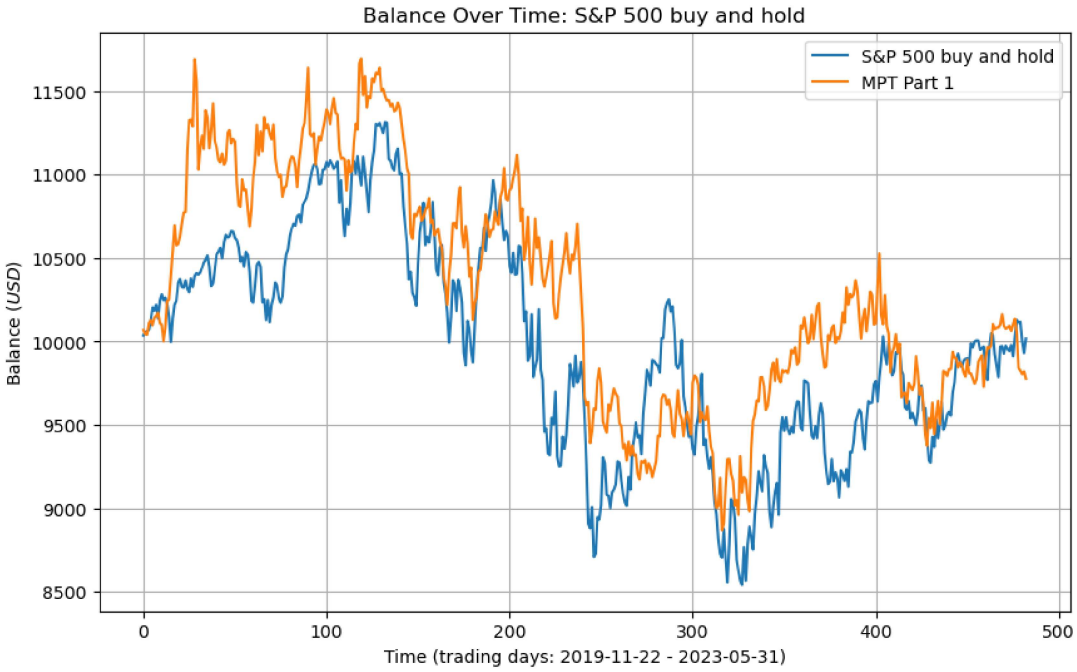

```
Number of winning trades:  241
Number of losing trades:  241
Total number of trades made:  482
Maximum equity drawdown for the whole portfolio:  24.5  %
```

17 Comparison

[60]:
```
# Create the plot
plt.figure(figsize=(10, 6))  # Adjust the figure size if needed
plt.plot(spy_data.index[len(Bal_vs_time_Part2):],
         spy_data.iloc[len(Bal_vs_time_Part2):],
         label='SPY Adj Close')
plt.xlabel('Date')
plt.ylabel('Adj Close')
plt.title('SPY Adj Close from 2019-11-22 to 2023-05-31')
plt.legend()
plt.grid(True)
plt.show()
```

```
plt.figure(figsize=(10, 6))  # Adjust the figure size if needed
time_values = list(range(len(Bal_vs_time_Part4)))
# Plot Bal_vs_time_Part5
plt.plot(time_values, Bal_vs_time_Part4, label='S&P 500 buy and hold')

# Plot Bal_vs_time_Part3
plt.plot(time_values, Bal_vs_time_Part1, label='MPT Part 1')

plt.xlabel('Time (trading days: 2019-11-22 - 2023-05-31)')
plt.ylabel('Balance ($USD$)')
plt.title('Balance Over Time: S&P 500 buy and hold')
plt.legend()
plt.grid(True)
plt.show()
```





MPT Part 1 and Part 2 results are very similar. These two variations (P1 using sample covariance, P2 using Ledoit-Wolf shrinkage) were produced in case there was issues of non-positive semi deifnite covariance matricies being present, the P1 MPT would miss the day of investing, whilst the P2 MPT using Ledoit-Wolf would overcome this issue, and allow for investing on that particular day. The maximum equity drawdown for our whole portfolio was calculated at: 24.2% during this trading period.

P3 MPT is also not feasible, adding optimisation constraints with L2 regularisation resulted in very few optimal portfolios being calculated, hence a significant number of trading days were missed. As when viewing post-processing result for P3, the number of trades made was 11 (11 trading days found), whilst P1 and P2 had 483 trading days. Additionally, in the graph above this text, this is 2 accounts both with a starting balance of £10,000, the orange graph represents an MPT with no L2 regularisation taking place and a daily rebalance period, whilst the blue graph represents a traditional buy-and-hold of the underlying asset we are attempting to outpreform: the S&P 500 index.

In summary, MPT is not a suitable strategy to generate significant returns. The sample covariance calculator experienced no issues with being non-positive semi definite, therefore an MPT with the sample covariance is suitable. L2 regularisation provided no benefits and actually reduced the number of trading oppurtunities available.