```python
# Importing and installing packages required for calculations and visualisation

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

!pip install numpy_financial
import numpy_financial as npf
```

```
Collecting numpy_financial
  Downloading numpy_financial-1.0.0-py3-none-any.whl (14 kB)
Requirement already satisfied: numpy>=1.15 in /shared-libs/python3.9/py/lib/python3.9/site-packages (from ı
Installing collected packages: numpy_financial
Successfully installed numpy_financial-1.0.0
WARNING: You are using pip version 22.0.4; however, version 23.2.1 is available.
You should consider upgrading via the '/root/venv/bin/python -m pip install --upgrade pip' command.
```

```python
!pip install pyfolio
import pyfolio as pf

!pip install scipy
from scipy import stats
from scipy.stats import norm

!pip install PyPortfolioOpt
from pypfopt.efficient_frontier import EfficientFrontier
from pypfopt import risk_models
from pypfopt import expected_returns
```

```
1.1/1.1 MB 52.7 MB/s eta 0:00:00
Requirement already satisfied: six>=1.5 in /shared-libs/python3.9/py-core/lib/python3.9/site-packages (fro
Installing collected packages: setuptools, scs, qdldl, ecos, osqp, cvxpy, PyPortfolioOpt
  Attempting uninstall: setuptools
    Found existing installation: setuptools 58.1.0
    Uninstalling setuptools-58.1.0:
      Successfully uninstalled setuptools-58.1.0
```

1. Craft an all-weather portfolio. Pick your own portfolio of stocks following an investment theme that has stood and will stand the test of time. Extract the data from Yahoo Finance/Refinitiv/any other platforms and explore the statistical distribution of each stock's past 5-year returns.

I have imported five ETFs as part of my all weather portfolio. The five ETFs comprise of: 1) Vanguard Total Stock Market Index Fund ETF **('VTI')**, 2) Invesco Optimum Yld Dvsfd Cmd Str No K 1 ETF **('PDBC')**, 3) Vanguard Intermediate-Term Treasury Index Fd ETF **('VGIT')**, 4) Vanguard Long-Term Treasury Index Fund ETF **('VGLT')** and 5) SPDR Gold MiniShares Trust **('GLDM')**.

The reason these ETFs were chosen are due to the following:

1) The ETFs are diversified across a range of industries. VTI, for example, has largest holdings in the technology sector (30%), but at the same time has exposure in key industries including consumer discretionary, financials and the healthcare sectors. On the other hand, GLDM is a primarily gold-backed ETF. It is interesting to test whether, and the extent to which having an exposure in both commodities and securities will allow stable returns in the event of market uncertainties.

2) The portfolio of ETF invests in both high risk and lower risk securities. For example, PDBC is a predominantly actively-managed fund investing in commodity-linked futures, while the VGIT and VGLT invests primarily in US treasury bonds. It would be interesting to test whether under market uncertainties, returns can still be generated by having a balanced exposure in both high and low risk securities.

3) Finally, when doing research on 'all weather portfolios', I came across a similar portfolio constructed by Ray Dalio, the founder of Bridgewater Associates, the largest hedge fund in the world. ([https://ofdollarsanddata.com/ray-dalio-all-weather-portfolio/)](https://ofdollarsanddata.com/ray-dalio-all-weather-portfolio/) He opined that having an exposure US bonds, stocks, gold and commodities would allow a portfolio to perform well under all market conditions. Hence, I would like to use this exercise to the the truthfulness of his statement.

```
# Importing the five ETFs forming the all weather portfolio, and indexing exchange dates

vti = pd.read_csv('VTI returns.csv', parse_dates = ['Exchange Date'], index_col = ['Exchange Date'])
pdbc = pd.read_csv('PDBC returns.csv', parse_dates = ['Exchange Date'], index_col = ['Exchange Date'
vgit = pd.read_csv('VGIT returns.csv', parse_dates = ['Exchange Date'], index_col = ['Exchange Date'
vglt = pd.read_csv('VGLT returns.csv', parse_dates = ['Exchange Date'], index_col = ['Exchange Date'
gldm = pd.read_csv('GLDM returns.csv', parse_dates = ['Exchange Date'], index_col = ['Exchange Date'
```

Run the app to see the outputs

Press the run button in the top right corner

```
# Visualising and checking first 5 rows of data

vti.head()
```

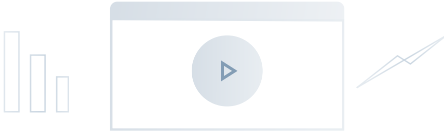
	Close float64	Adjusted Close flo...	Net float64	%Chg object	Open float64	Low float64
20...	223.57	224.68	-1.11	-0.49%	224.26	223.3
20...	224.68	nan	0.74	+0.33%	225.34	223.9
20...	223.94	nan	-0.24	-0.11%	224.5	223.8
20...	224.18	nan	0.99	+0.44%	223.25	223.0
20...	223.19	nan	3.2	+1.45%	219.94	219.

5 rows, showing 10 per page Page 1 of 1

```
# Joining the closing prices for the 5 ETFs according to respective exchange dates

column_stock = ['VTI', 'PDBC', 'VGIT', 'VGLT', 'GLDM']

combined_closing_prices = pd.concat([vti[['Close']], pdbc[['Close']], vgit[['Close']], vglt[['Close'

combined_closing_prices.columns = column_stock

print(combined_closing_prices)
```

```
                  VTI   PDBC   VGIT   VGLT   GLDM
Exchange Date
2018-09-06     148.92  17.87  62.59  73.83  23.98
2018-09-07     148.58  17.92  62.38  73.23  23.92
2018-09-10     148.90  17.96  62.37  73.48  23.90
2018-09-11     149.40  18.14  62.25  72.97  23.92
2018-09-12     149.41  18.28  62.29  73.13  24.11
...               ...    ...    ...    ...    ...
2023-08-29     223.19  14.66  58.15  59.86  38.46
2023-08-30     224.18  14.70  58.13  59.82  38.56
2023-08-31     223.94  14.75  58.26  60.07  38.48
2023-09-01     224.68  14.93  57.92  59.00  38.51
2023-09-05     223.57    NaN    NaN    NaN    NaN

[1257 rows x 5 columns]
```

```
# Cleaning data - dropping the last row showing exchange date data for only one stock (5th Sep 2023)

date_to_drop = pd.to_datetime('2023-09-05')
combined_closing_prices2 = combined_closing_prices.drop(date_to_drop)
```



Run the app to see the outputs

```python
# Visualising cleaned data

combined_closing_prices2
```

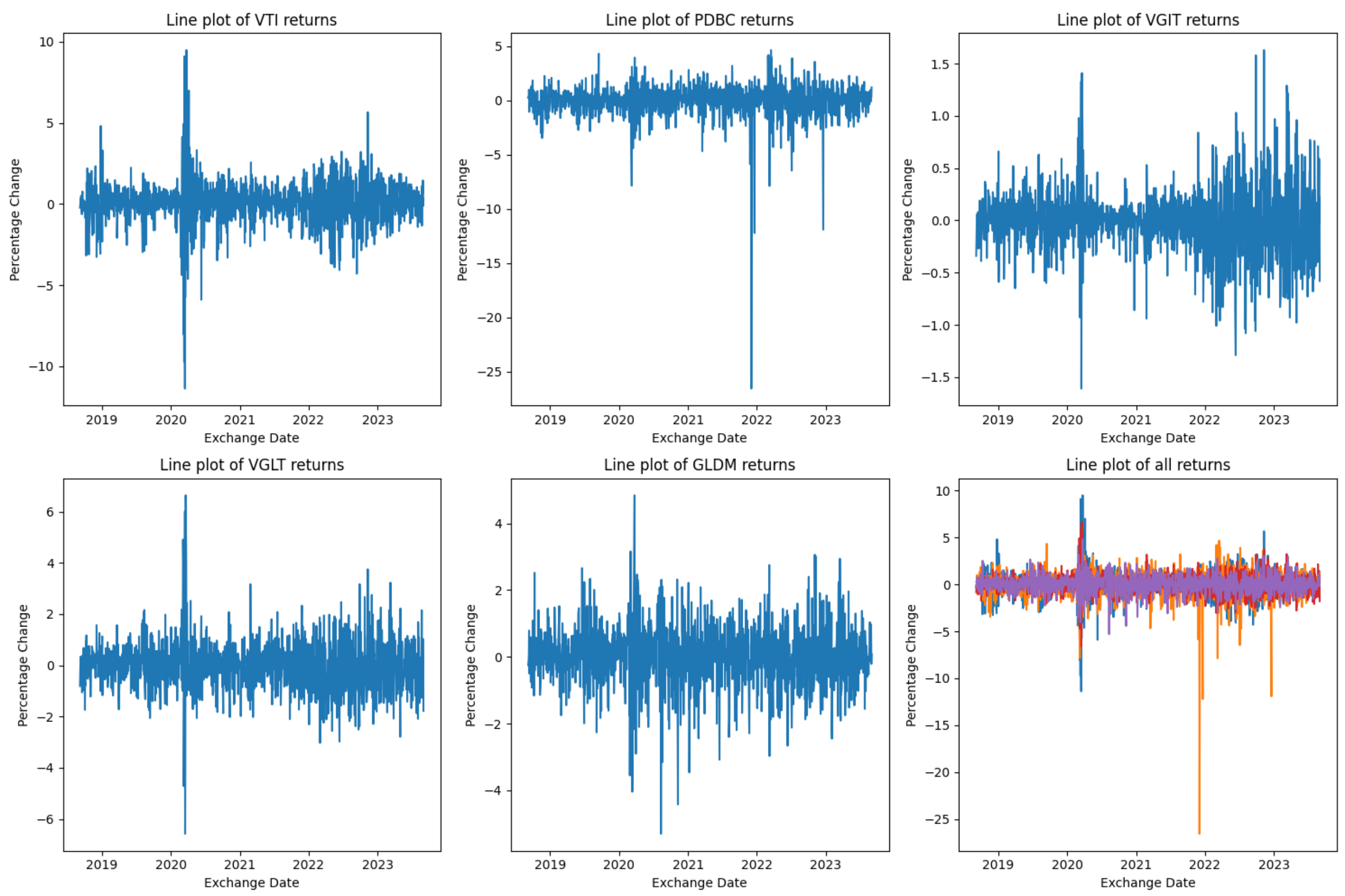
	VTI float64 111.91 - 242.97	PDBC float64 11.22 - 22.71	VGIT float64 57.14 - 70.85	VGLT float64 57.38 - 105.03	GLDM float64 23.68 - 41.14
20...	218.59	14.63	57.7	59.18	37.97
20...	219.99	14.62	57.81	59.25	38.09
20...	223.19	14.66	58.15	59.86	38.46
20...	224.18	14.7	58.13	59.82	38.56
20...	223.94	14.75	58.26	60.07	38.48
20...	224.68	14.93	57.92	59	38.51

1256 rows, showing 10 per page   Page 1 of 126

```python
# Exploring return data of individual stocks (ETFs)

# Creating multiple figures to plot return data
fig, ax = plt.subplots(nrows = 2, ncols = 3, sharex =False, sharey =False, figsize = (15, 10))

# Creating a for loop to describe statistical data and plot line plot of each return data
for i, name in enumerate(combined_closing_prices2.columns):
    row, col = divmod(i, 3)
    ax[row, col].plot(round(combined_closing_prices2[name].pct_change()*100, 2))
    ax[row, col].set_title('Line plot of ' + name + ' returns')
    ax[row, col].set_xlabel('Exchange Date')
    ax[row, col].set_ylabel('Percentage Change')
    print('Description of Statistical Data of ' + str(name) + '\n' + str(combined_closing_prices2[nar

# Creating a for loop to plot all return data into one graph
for name in combined_closing_prices2.columns:
    all_returns1 = round(combined_closing_prices2[name].pct_change()*100, 2)
    ax[1,2].plot(all_returns1)
    ax[1,2].set_title('Line plot of all returns')
    ax[1,2].set_xlabel('Exchange Date')
    ax[1,2].set_ylabel('Percentage Change')

plt.tight_layout()  # Automatically adjust subplot parameters to fit the figure area
plt.show()
```

```
Description of Statistical Data of VTI
count    1256.000000
mean      185.098623
std        32.875156
min       111.910000
25%       152.015000
50%       191.585000
```

```
75%        214.285000
max        242.970000
Name: VTI, dtype: float64

Description of Statistical Data of PDBC
count    1256.000000
mean       16.243400
std         2.351572
min        11.220000
25%        14.457500
50%        16.130000
75%        17.562500
max        22.710000
Name: PDBC, dtype: float64

Description of Statistical Data of VGIT
count    1256.000000
mean       64.846170
std         4.000308
min        57.140000
25%        61.455000
50%        65.770000
```



The 6 graphs above show the percentage returns of the respective ETFs' across the 5 years where price data was collected. There are several conclusions that can be drawn - firstly, that the returns across the ETFs are, generally speaking, relatively stable. The returns hover around a band of between -5% to 5%. The ETFs that have greater volatility in prices appear to be PDBC and GLDM. - the ETFs which invest in commodities and gold. This is perhaps unsurprising, given that the commodities industries have traditionally offered superior returns, but at the same time comes with greater standard deviation (risk).

```python
# Calculating mean return for the 5 ETFs

returns = combined_closing_prices2.pct_change()

# Drop any NaN values

returns = returns.dropna()

returns
```

	VTI float64 -0.1138085779445...	PDBC float64 -0.2656481025135...	VGIT float64 -0.0161313347608...	VGLT float64 -0.0657711442786...	GLDM float64 -0.053049082796...
201...	-0.002283105023	0.00279798545	-0.003355168557	-0.008126777733	-0.002502085071
201...	0.002153721901	0.002232142857	-0.000160307791	0.003413901407	-0.0008361204013
201...	0.003357958361	0.01002227171	-0.001924001924	-0.006940664126	0.0008368200837
201...	0.000066934404..	0.007717750827	0.0006425702811	0.002192681924	0.007943143813
201...	0.005019744328	-0.01039387309	0	0.001093942295	-0.004562422231
201...	0.0007991475759	-0.005527915976	-0.001926472949	-0.00437098757	-0.005
201...	-0.006321533138	-0.001667593107	0.0008042464211	0.0002743860612	0.005862646566
201...	0.005357262439	0.008351893096	-0.002732240437	-0.01056096557	-0.002497918401
201...	0.0001332178778	0.006626173385	-0.0008058017728	-0.004990296645	0.004173622705
201...	0.007725607726	0	0.0003225806452	0.003761493452	0.003325020781

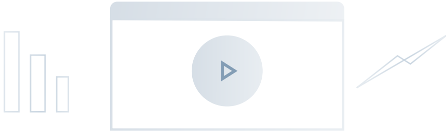
1255 rows, showing 10 per page   Page 1 of 126

```python
# Statistics for the returns of the 5 ETFs

returns.describe()
```

	VTI float64	PDBC float64	VGIT float64	VGLT float64	GLDM float64
cou...	1255	1255	1255	1255	1255
me...	0.0004243340496	-0.00001637882...	-0.00005669653...	-0.0001291219558	0.0004202465483
std	0.01386503374	0.01543909953	0.003192014987	0.009963169618	0.009241103588
min	-0.1138085779	-0.2656481025	-0.01613133476	-0.06577114428	-0.0530490828
25%	-0.005567126616	-0.005930222153	-0.001623256852	-0.006153132246	-0.004069384163
50%	0.0007991475759	0.001413427562	0	0.0001045806317	0.0006361323155
75%	0.007358058445	0.007604856768	0.001508183336	0.005570812747	0.005471837847
max	0.09489768564	0.04655380895	0.01630246271	0.06636633337	0.04838709677

8 rows, showing 10 per page   Page 1 of 1

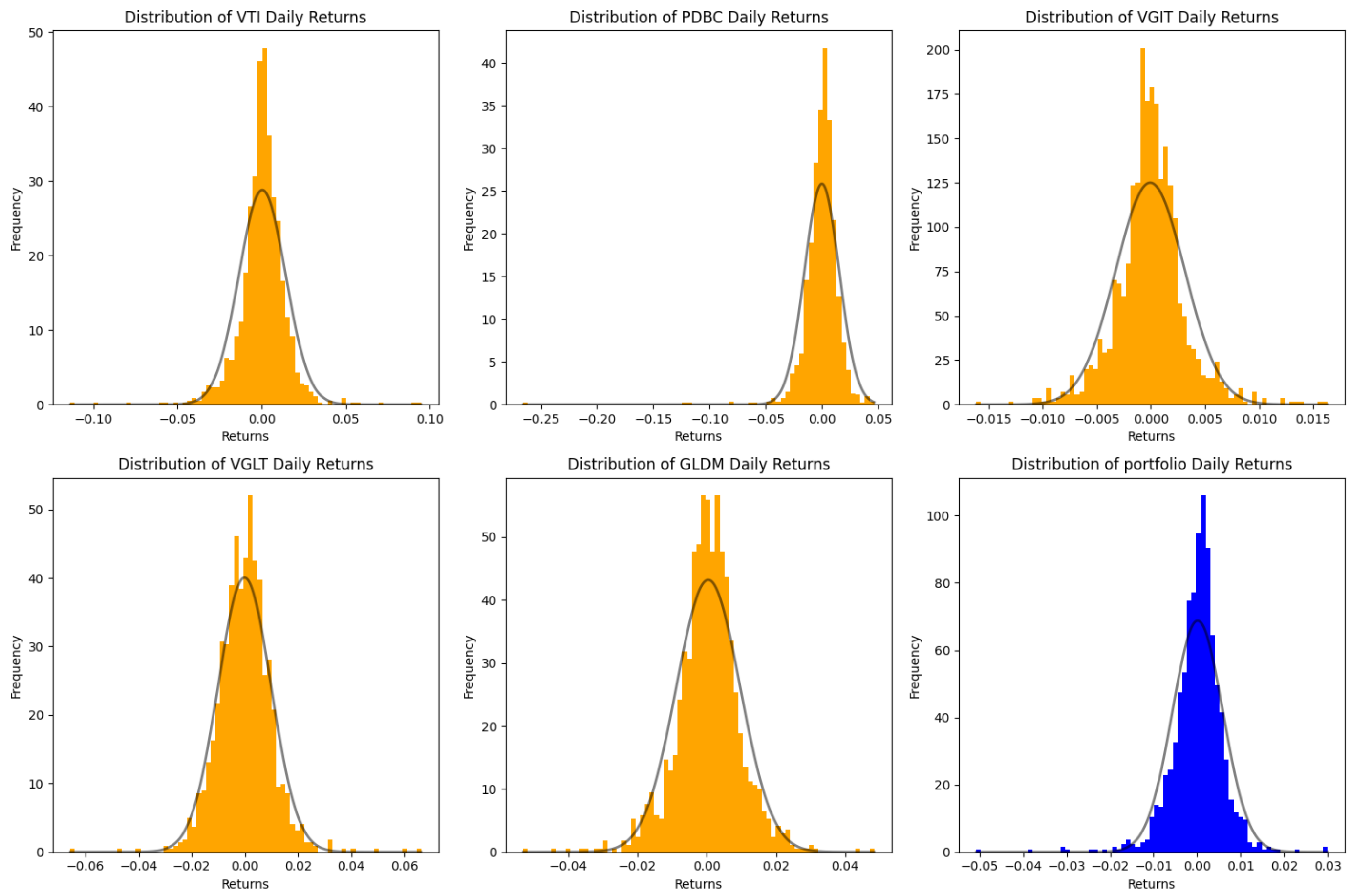
```python
# Calculating portfolio returns - factoring in equal weights

weights = np.array([0.2, 0.2, 0.2, 0.2, 0.2])

portfolio_returns = returns.dot(weights)

returns['portfolio'] = portfolio_returns
```



Run the app to see the outputs

Press the run button in the top right corner

```python
# Creating a for loop to show distribution returns for the 5 ETFs

# Creating subplots prior to executing the for loop

fig, ax = plt.subplots(nrows = 2, ncols = 3, sharex =False, sharey =False, figsize = (15, 10))

# Creating for loop to plot distribution of each ETF return in the portfolio

for i, name in enumerate(returns.columns):
    row, col = divmod(i, 3)  # Calculate the row and column indices
    ax[row, col].hist(returns[name], bins=75, color="orange", density = True)
    ax[row, col].set_title('Distribution of ' + name + ' Daily Returns')
    ax[row, col].set_xlabel('Returns')
    ax[row, col].set_ylabel('Frequency')

    # Plot the fitted normal distribution
    xmin = returns[name].min()
    xmax = returns[name].max()
    mu, std = norm.fit(returns[name])
    x = np.linspace(xmin, xmax, 1255)  # Increased the number of points for smoother plot
    p = norm.pdf(x, mu, std)
    ax[row, col].plot(x, p, color="black", linewidth = 2, alpha=0.5)

# Plotting the portfolio returns into subplots
ax[1,2].hist(returns['portfolio'], bins = 75, color = 'blue', density = True)

plt.tight_layout()  # Automatically adjust subplot parameters to fit the figure area
plt.show()
```




```python
kurt = []

# Creating a for loop to calculate skewness and kurtosis of each ETF, as well as the portfolio

for name in returns.columns:
    skew.append(returns[name].skew())
    kurt.append(returns[name].kurtosis())

# Placing the skew, kurt values into the skew_kurt dataframe for easier visualization

skew_kurt = pd.DataFrame({'Skew': skew, 'Kurtosis': kurt}, index = stock_names1)

skew_kurt
```

	Skew float64	Kurtosis float64
VTI	-0.5063729893	10.86504167
PD...	-5.19303569	76.25044743
VGIT	0.2058752889	3.204608297
VGLT	0.2673715326	5.019850954
GL...	-0.2652276957	3.116287954
por...	-1.150642383	9.38399308

6 rows, showing 10 per page Page 1 of 1

The above distributions, in addition to the kurtosis and skewness data demonstrates that the ETF returns for particular stocks are high. For example, PDBC, which primarily invests in commodities including gasoline, crude oil, sugar and various metals. While the skew and kurtosis values are extremely high, this is perhaps unsurprising given the volatility of the ETF. For example, there are numerous occasions that PDBC has increased or decreased 10% in value in one trading day, with the greatest change in December 2021, when the ETF lost close to 27% of its value in one day.

That said, the effect of diversification can be seen in the significantly lower portfolio skewness and kurtosis.

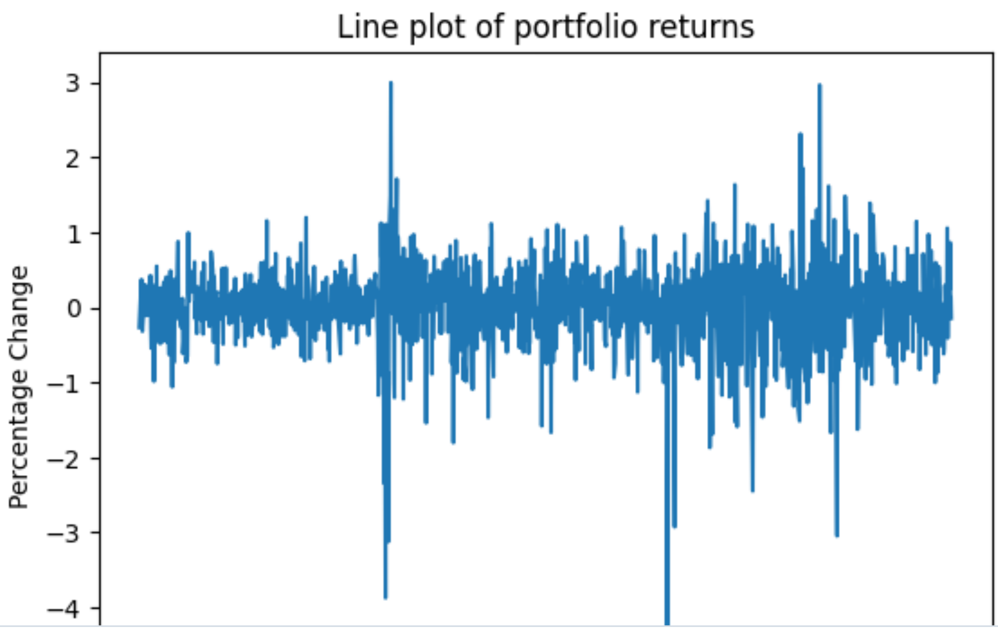
```
# Assigned weights in Part 1 and reprinting an equally weighted portfolio

returns
```

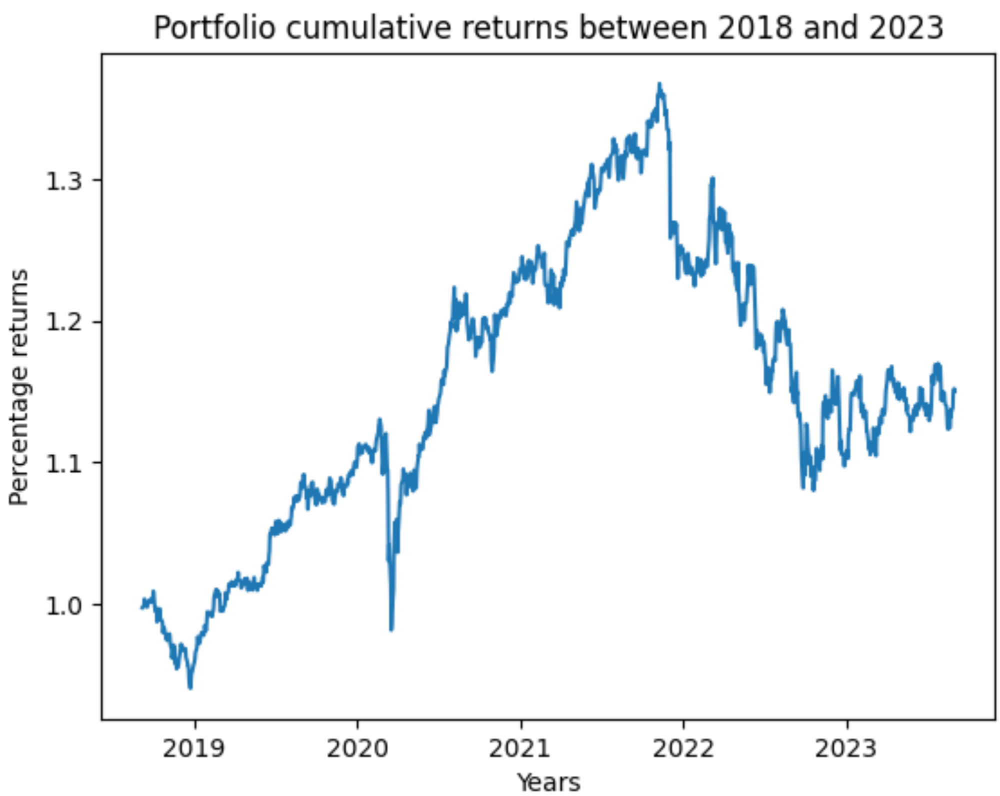
	VTI float64	PDBC float64	VGIT float64	VGLT float64	GLDM float64	portfolio float64
	-0.1138085779445...	-0.2656481025135...	-0.0161313347608...	-0.0657711442786...	-0.053049082796...	-0.0507527742033
201...	-0.002283105023	0.00279798545	-0.003355168557	-0.008126777733	-0.002502085071	-0.00269383018
201...	0.002153721901	0.002232142857	-0.000160307791	0.003413901407	-0.0008361204013	0.00136066759
201...	0.003357958361	0.01002227171	-0.001924001924	-0.006940664126	0.0008368200837	0.00107047682
201...	0.000066934404..	0.007717750827	0.0006425702811	0.002192681924	0.007943143813	0.0037126162
201...	0.005019744328	-0.01039387309	0	0.001093942295	-0.004562422231	-0.00176852173
201...	0.0007991475759	-0.005527915976	-0.001926472949	-0.00437098757	-0.005	-0.00320524578
201...	-0.006321533138	-0.001667593107	0.0008042464211	0.0002743860612	0.005862646566	-0.00020956943
201...	0.005357262439	0.008351893096	-0.002732240437	-0.01056096557	-0.002497918401	-0.000416393775
201...	0.0001332178778	0.006626173385	-0.0008058017728	-0.004990296645	0.004173622705	0.001027383
201...	0.007725607726	0	0.0003225806452	0.003761493452	0.003325020781	0.0030269405

1255 rows, showing 10 per page · Page 1 of 126

```
# Visualising portfolio returns through line plot

plt.plot(returns.portfolio*100)

plt.xlabel('Exchange Date')

plt.ylabel('Percentage Change')

plt.title('Line plot of portfolio returns')
```

```
Text(0.5, 1.0, 'Line plot of portfolio returns')
```






```
# Pulling the closing prices of the respective ETFs before analysing the risk and return of stocks

combined_closing_prices2
```

	VTI float64 111.91 - 242.97	PDBC float64 11.22 - 22.71	VGIT float64 57.14 - 70.85	VGLT float64 57.38 - 105.03	GLDM float64 23.68 - 41.14
201...	148.92	17.87	62.59	73.83	23.98
201...	148.58	17.92	62.38	73.23	23.92
201...	148.9	17.96	62.37	73.48	23.9
201...	149.4	18.14	62.25	72.97	23.92
201...	149.41	18.28	62.29	73.13	24.11

201...	150.16	18.09	62.29	73.21	24	
201...	150.28	17.99	62.17	72.89	23.88	
201...	149.33	17.96	62.22	72.91	24.02	
201...	150.13	18.11	62.05	72.14	23.96	
201...	150.15	18.23	62	71.78	24.06	

1256 rows, showing 10 per page Page 1 of 126

Risk and return of portfolio

```
# Total return of portfolio - calculated using for loop

total_return_per_stock = []

# Creating a for loop to append total return for each ETF into total_return_per_stock list

for name in combined_closing_prices2.columns:
    total_return_per_stock.append((combined_closing_prices2[name][-1] - combined_closing_prices2[name

total_return_per_stock_df = pd.DataFrame({'Returns': total_return_per_stock}, index = ['VTI', 'PDBC',

# Total return of portfolio from individual stock returns

total_return_portfolio = np.sum(total_return_per_stock*weights.T)

# Annualised returns of portfolio

annualised_return = ((1+total_return_portfolio)**(1/5)) - 1

# Volatility of portfolio

vol_port = returns.portfolio.std() * np.sqrt(250)

# Sharpe ratio - assuming risk free rate is 4.27% (according to 10 year US treasury rate as of 7 Sep

rfr = 0.0427
port_sharpe_ratio = ((annualised_return - rfr) / vol_port)

# Sortino ratio of portfolio

target_rate = 0
downside_returns = returns.loc[returns['portfolio'] < target_rate]

expected_return = returns['portfolio'].mean()
down_stdev = downside_returns['portfolio'].std()

sortino_ratio = (expected_return - rfr)/down_stdev

#Summarizing findings into a table for easier visualisation

summary1 = pd.DataFrame({'Total Return': total_return_portfolio, 'Annualised Return': annualised_retu

summary1
```

	Total Return float64	Annualised Return f...	Volatility float64	Sharpe Ratio float64	Sortino Ratio float64
Por...	0.1349300324	0.02563732588	0.09176574226	-0.1859372975	-8.663312509

1 row, showing 10 per page Page 1 of 1

The data above shown in the dataframe highlights that the portfolio has not performed exceptionally well in the past 5 years. It has generated total returns of 13.4%, equivalent to an annual return of around 2.5% This, in light of its relatively high volatility (9.2%) does not appear to be a very successful portfolio. Its negative sortino ratio of -8.6 shows that the portfolio has generated returns lower than the risk free rate (assumed to be 4.27%) - indicating that downside volatility (risk of losses) has been high compared to the returns achieved.

3. Repeat (2) but with the maximum Sharpe portfolio and minimum volatility portfolio.

```
# Calculating portfolio mean, volatility, skewness and kurtosis

portfolio_mean = returns.portfolio.mean()

portfolio_volatility = returns.portfolio.std() * np.sqrt(250)

portfolio_skewness = returns.portfolio.skew()

portfolio_kurtosis = returns.portfolio.kurt()

print('Portfolio Mean: ' + str(portfolio_mean) + '\n' + 'Portfolio Volatility: ' + str(portfolio_vola
```

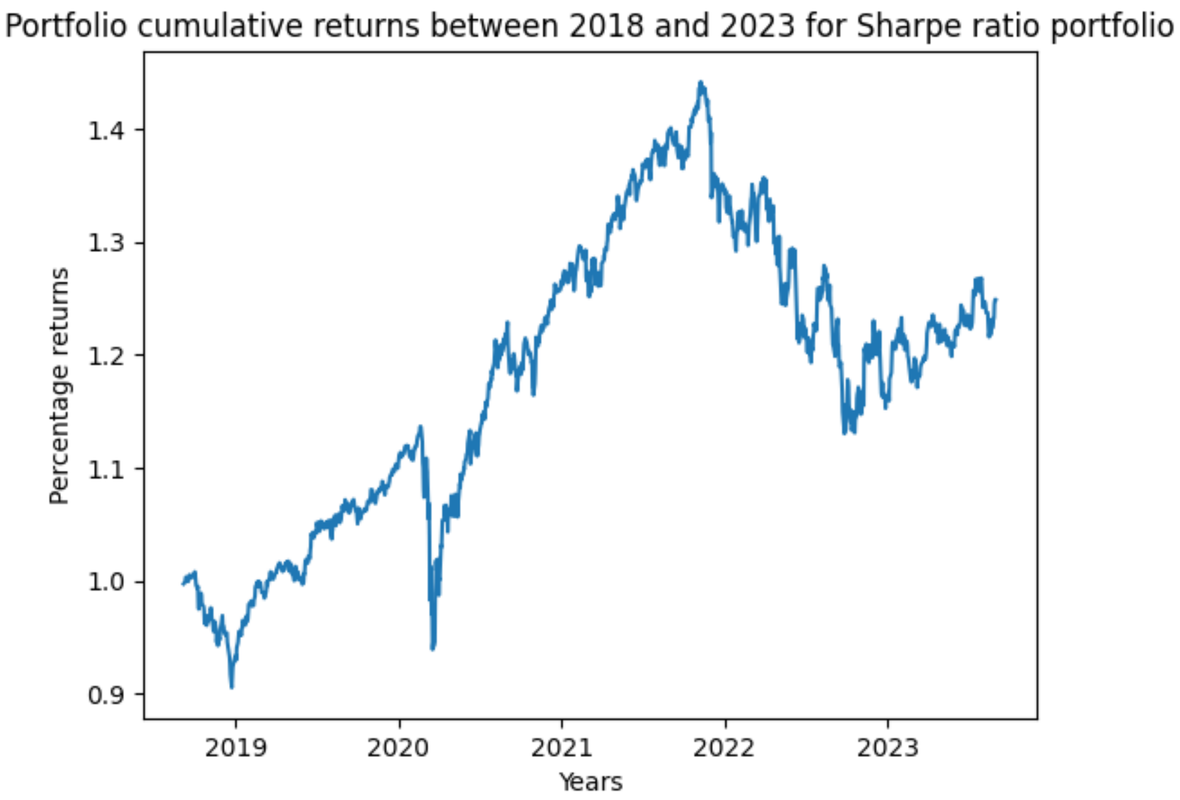
```
Portfolio Mean: 0.00012847665681194108
Portfolio Volatility: 0.09176574225621523
Portfolio Skewness: -1.150642382601818
Portfolio Kurtosis: 9.383993079951
```

```
# Define parameters for Efficient Frontier

mu = expected_returns.mean_historical_return(combined_closing_prices2)
Sigma = risk_models.sample_cov(combined_closing_prices2)

# Define the efficient frontier

ef = EfficientFrontier(mu, Sigma)

# Calculate weights for the maximum Sharpe portfolio

raw_weights_maxsharpe = ef.max_sharpe()
cleaned_weights_maxsharpe = ef.clean_weights()

print('Weights:', cleaned_weights_maxsharpe)
MaxSharpe = ef.portfolio_performance(verbose = True)
```

```
Weights: OrderedDict([('VTI', 0.24517), ('PDBC', 0.0), ('VGIT', 0.0), ('VGLT', 0.0), ('GLDM', 0.75483)])
Expected annual return: 9.6%
Annual volatility: 12.7%
Sharpe Ratio: 0.60
```

The maximum Sharpe ratio of 0.60 indicates that the portfolio is able to provide a reasonable level of return for the risk taken, beyond the risk free rate. However, to achieve this Sharpe ratio, the portfolio will only invest in two stocks - VTI and GLDM. To have a better understanding of the Sharpe ratio, I have plotted the cumulative returns of this portfolio below.

```
# Weights for the sharpe ratio portfolio

sharpe_weights = np.array([0.24517, 0.75483])

VTI2 = returns.iloc[:, 0]  # Select the first column
GLDM2 = returns.iloc[:, -1]    # Select the last column

# Form a new DataFrame with the selected columns
sharpe_portfolio = pd.DataFrame({'VTI': VTI2, 'GLDM': GLDM2})

sharpe_returns = sharpe_portfolio.dot(sharpe_weights)

sharpe_portfolio['portfolio'] = sharpe_returns

sharpe_portfolio

# Calculating and visualising cumulative returns for the portfolio, utilising Sharpe ratio calculati

cumulative_returns_sharpe = (1+sharpe_portfolio.portfolio).cumprod()

plt.xlabel('Years')

plt.ylabel('Percentage returns')

plt.title('Portfolio cumulative returns between 2018 and 2023 for Sharpe ratio portfolio')

plt.plot(cumulative_returns_sharpe)
plt.show()
```

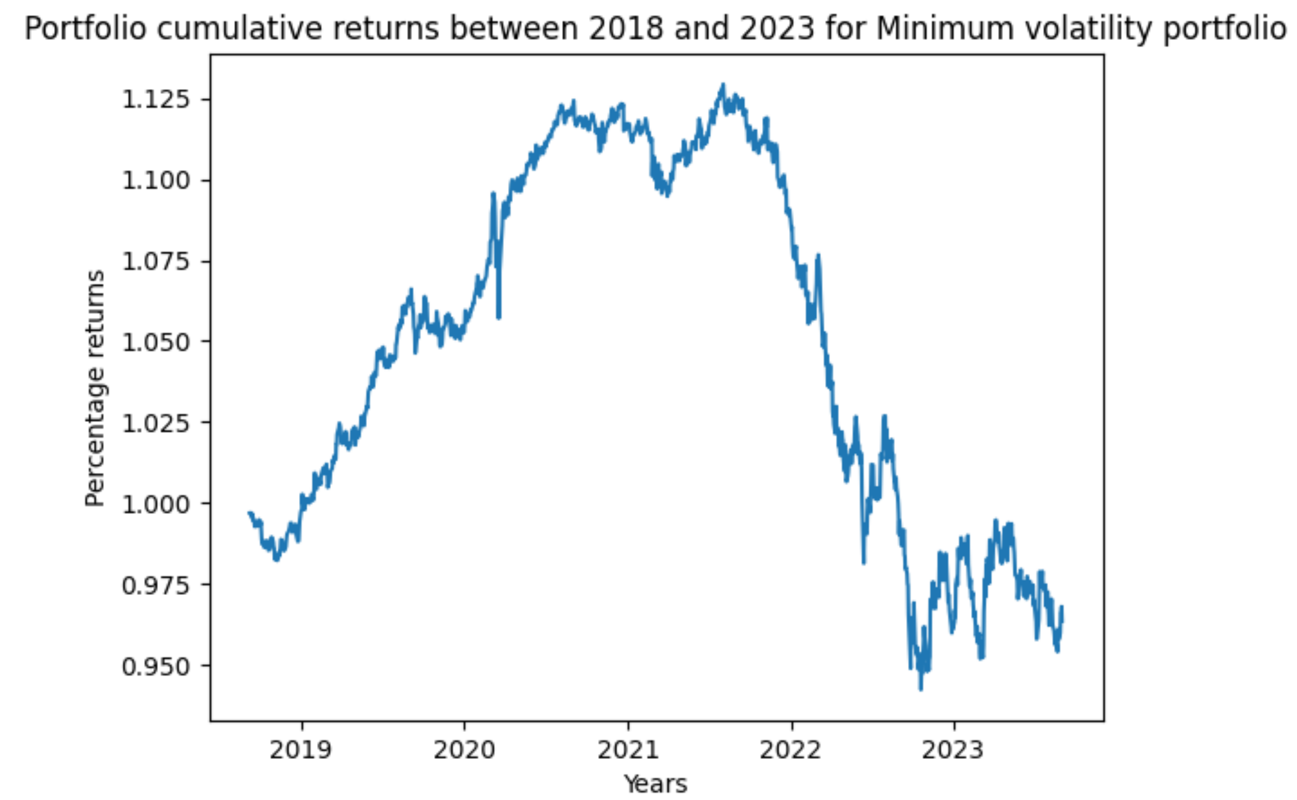


Accounting for the weights provided by the Sharpe ratio calculations, there are a few alarming factors. Firstly, the portfolio appears to be able to generate higher returns (exceeding 20% as of Sep 2023, compared to 13% for the equal weighted portfolio). During the peak of both portfolio's performance, the Sharpe ratio portfolio has already generated returns exceeding 40%, which is superior than that of the equal weighted portfolio.

```
# Calculate weights for the minimum volatility portfolio

# Define the efficient frontier

ef1 = EfficientFrontier(mu, Sigma)

# Calculate weights for the minimum volatility portfolio

raw_weights_minvol = ef1.min_volatility()
cleaned_weights_minvol = ef1.clean_weights()

print('Weights:', cleaned_weights_minvol)
MinVol = ef1.portfolio_performance(verbose = True)
```

```
Weights: OrderedDict([('VTI', 0.0624), ('PDBC', 0.03597), ('VGIT', 0.90163), ('VGLT', 0.0), ('GLDM', 0.0)]
Expected annual return: -1.0%
Annual volatility: 4.7%
Sharpe Ratio: -0.64
```

```
# Weights for the minimum volatility portfolio

minvol_weights = np.array([0.0624, 0.03597, 0.90163])

VTI3 = returns.iloc[:, 0]  # Select the first column
PDBC3 = returns.iloc[:, 1]   # Select the second column
VGIT3 = returns.iloc[:, 2]   # Select the last  column

# Form a new DataFrame with the selected columns
minvol_portfolio = pd.DataFrame({'VTI': VTI3, 'PDBC': PDBC3, 'VGIT': VGIT3})

minvol_returns = minvol_portfolio.dot(minvol_weights)

minvol_portfolio['portfolio'] = minvol_returns

# Calculating and visualising cumulative returns for the portfolio, utilising Sharpe ratio calculati

cumulative_returns_minvol = (1+minvol_portfolio.portfolio).cumprod()

plt.xlabel('Years')

plt.ylabel('Percentage returns')

plt.title('Portfolio cumulative returns between 2018 and 2023 for Minimum volatility portfolio')

plt.plot(cumulative_returns_minvol)
plt.show()
```



seen by the graph above, cumulative returns has dropped over 20%), in addition to the fall of commodity prices in the past 1-2 years, which makes up a significant part of my portfolio.

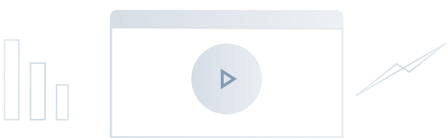
4. [Optional Bonus] With PyPortfolioOpt, how can you further improve your all-weather portfolio construction?

```
# Installing relevant packages

!python -m pip install --upgrade pip
!pip install PyPortfolioOpt
!pip install pypfopt
from pypfopt.black_litterman import BlackLittermanModel
from pypfopt import objective_functions
```

```
Requirement already satisfied: pip in /root/venv/lib/python3.9/site-packages (22.0.4)
Collecting pip
  Downloading pip-23.2.1-py3-none-any.whl (2.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 20.6 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 22.0.4
    Uninstalling pip-22.0.4:
      Successfully uninstalled pip-22.0.4
Successfully installed pip-23.2.1
Requirement already satisfied: PyPortfolioOpt in /root/venv/lib/python3.9/site-packages (1.5.5)
Requirement already satisfied: cvxpy<2.0.0,>=1.1.19 in /root/venv/lib/python3.9/site-packages (from PyPort
Requirement already satisfied: numpy<2.0.0,>=1.22.4 in /shared-libs/python3.9/py/lib/python3.9/site-packag
Requirement already satisfied: pandas>=0.19 in /shared-libs/python3.9/py/lib/python3.9/site-packages (from
```

```
Requirement already satisfied: scipy<2.0,>=1.3 in /shared-libs/python3.9/py/lib/python3.9/site-packages (fr
Requirement already satisfied: osqp>=0.4.1 in /root/venv/lib/python3.9/site-packages (from cvxpy<2.0.0,>=1
Requirement already satisfied: ecos>=2 in /root/venv/lib/python3.9/site-packages (from cvxpy<2.0.0,>=1.1.19
Requirement already satisfied: scs>=1.1.6 in /root/venv/lib/python3.9/site-packages (from cvxpy<2.0.0,>=1.1
Requirement already satisfied: setuptools>65.5.1 in /root/venv/lib/python3.9/site-packages (from cvxpy<2.0
Requirement already satisfied: python-dateutil>=2.7.3 in /shared-libs/python3.9/py-core/lib/python3.9/site-
Requirement already satisfied: pytz>=2017.3 in /shared-libs/python3.9/py/lib/python3.9/site-packages (from
Requirement already satisfied: qdldl in /root/venv/lib/python3.9/site-packages (from osqp>=0.4.1->cvxpy<2.0
Requirement already satisfied: six>=1.5 in /shared-libs/python3.9/py-core/lib/python3.9/site-packages (from
ERROR: Could not find a version that satisfies the requirement pypfopt (from versions: none)
ERROR: No matching distribution found for pypfopt
```

```
# Calculating an exponentially moving average return, and sample covariance matrix

mu_ema = expected_returns.ema_historical_return(combined_closing_prices2, span=252, frequency=252)
sigma_ew = risk_models.exp_cov(combined_closing_prices2, span = 252, frequency = 252)

# Black-Litterman model

'''
Stock forecasts from online sources
VTI: +8%
PDBC: +4.5%
VGIT: -5%
VGLT: -10%
GLDM: +20%

'''

# Creating dictionary to show market prediction on the ETFs
marketview = {'VTI': 0.08, 'PDBC': 0.045, 'VGIT': -0.05, 'VGLT': -0.1, 'GLDM': 0.2}

from pypfopt import BlackLittermanModel, expected_returns, risk_models, EfficientFrontier, objective_

# Create BlackLittermanModel
bl = BlackLittermanModel(sigma_ew, pi=mu_ema, absolute_views=marketview)

# Calculate Black-Litterman returns and covariance matrix
ret_bl = bl.bl_returns()
S_bl = bl.bl_cov()
```

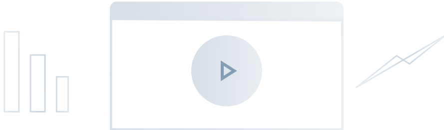


Run the app to see the outputs

Press the run button in the top right corner

```
# Create EfficientFrontier object with the returns and covariance matrix
ef2 = EfficientFrontier(ret_bl, S_bl)

# Adding L2 regularisation
ef2.add_objective(objective_functions.L2_reg, gamma=0.1)

# Compute the weights for the minimum volatility portfolio
raw_weights_bl = ef2.min_volatility()
```

```
# Clean the weights
cleaned_weights_bl = ef2.clean_weights()

weights_df = pd.DataFrame(cleaned_weights_bl, index = ["weights"])
weights_array = weights_df.values[0]
```



Run the app to see the outputs

Press the run button in the top right corner

```
# Extract weights from model

weights_array_df = pd.DataFrame(weights_array, index = ["VTI", "PDBC", "VGIT", "VGLT", "GLDM"], colum
model = ef2.portfolio_performance(verbose=True)
```

```
Expected annual return: 2.6%
Annual volatility: 9.0%
Sharpe Ratio: 0.06
```

```
weights_array_df
```

	weights float64
VTI	0.19757
PD...	0.17192
VGIT	0.24289
VGLT	0.18632
GL...	0.2013

5 rows, showing 10 per page Page 1 of 1

Reflection for Q4

The weights after incorporating PyPortfolioOpt weights still do not look ideal, and is inferior to the expected returns and Sharpe ratio figures calculated as part of question 3. I think an interesting topic to explore would be to test whether gamma values, number of stock in portfolios, as well as the market prediction on ETF growth or fall would change the performance of this black-litterman model, and if so, the extent to which these factors affect its performance.