Programming for Financial Data Science
Assignment 3: 4th April, 2023

Author: Gavin Connolly
Student Number: 18308483

Module: FIN42120 - Programming for Financial Data Science

Academic Year - 2022/23
Submitted: 4th April, 2023

Q1:

Create a Python DataFrame of simple returns of the 4 ETF.

We create a dataframe of historical weekly returns by iterating through a list containing the tickers of each of the 4 ETFs. We download weekly prices of each ETF, before calculating simple returns using the adjusted close price (Adj Close) and add these returns to the columns an empty dataframe df_hist. We then drop the first row from the dataframe as by calculating the returns this row will be filled by NaN's.

```
import pandas as pd
import yfinance as yf
import numpy as np
from scipy.stats import norm

tickers_list = ['IVV', 'IEUR', 'AIA', 'IEMG']

df_hist = pd.DataFrame()
for ticker_name in tickers_list:
    hist = yf.download(ticker_name, start="2015-01-01", end="2022-12-31",  interval="1wk", progress=False)
    hist['Returns'] = hist['Adj Close'].pct_change()
    df_hist[ticker_name] = hist['Returns']
df_hist.drop(df_hist.index[0], inplace=True)
```

Q2:

Write a Python function to generate the following summary statistics for each of the 4 ETFs: Mean, Median, Standard deviation, Skewness, Kurtosis, and Sharpe ratio. For this and subsequent calculations, assume a risk-free rate of 0. Test your function using the data item 1. above.

We calculate each of the summary statistics using the corresponding pandas functions. We then concatenate each of the statistics to an output dataframe & perform some simple formatting, before outputting our summary dataframe.

```
def returns_summary(returns, riskfree_rate=0):

    mean_return     = returns.mean()
    median_return   = returns.median()
    sd_retrurns     = returns.std()
    skew_returns    = returns.skew()
    kurt_returns    = returns.kurt()
    sharpe_returns  = (mean_return-riskfree_rate)/sd_retrurns

    summary = pd.concat([mean_return, median_return, sd_retrurns, skew_returns, kurt_returns, sharpe_returns], axis=1)
    summary.columns = ['Mean', 'Median', 'Std. Dev.', 'Skewness', 'Kurtosis', 'Sharpe Ratio']
    return summary

print(f'Summary of ETF Returns:\n{returns_summary(df_hist).to_latex(float_format="%.4f")}')
```

This provides the following summary statistics for each of the corresponding returns figures:

ETF	Mean	Median	Std. Dev.	Skewness	Kurtosis	Sharpe Ratio
IVV	0.0021	0.0042	0.0231	-1.1432	6.1540	0.0928
IEUR	0.0012	0.0042	0.0273	-1.2788	9.6524	0.0450
AIA	0.0013	0.0031	0.0301	-0.1010	2.0831	0.0435
IEMG	0.0009	0.0040	0.0277	-0.7962	4.6620	0.0321

Q3:

Write a Python function to calculate the Bayes-Stein estimates of the Mean and Covariance matrix of the return of a portfolio of the 4 assets. Test your function using the returns data in 1. above.

We define our Bayes-Stein estimation formula for a dataframe of returns as follows. We also call a function to calculate the weights of the Global Minimum Variance Portfolio which is outlined in the code section in Q4. Care is taken to ensure the dimensions of the mean returns vector & ones vector are of the correct defined, explicitly converting each to a column vector.

```
def bayes_stein(returns):

    T, K = returns.shape
    ones = np.ones((K,1))
    mu = np.array(returns.mean()).reshape((K,1))
    sigma = returns.cov()

    w_mv = minvar_weights(returns)
    mu_mv = w_mv.T @ mu

    lamb = float(np.divide((K+2)*(T-1)/(T-K-2), (mu-mu_mv * ones).T @ np.linalg.inv(sigma) @ (mu-mu_mv * ones)))
    gamma = lamb/(T+lamb)

    mu_bs = gamma * ones * mu_mv + (1-gamma) * mu
    sigma_bs = sigma * (1 + 1/(T+lamb)) + lamb/(T*(T+1+lamb)) * (ones @ ones.T /(ones.T @ np.linalg.inv(sigma) @ ones))
    return mu_bs, sigma_bs

mu_bs, sigma_bs = bayes_stein(df_hist)
print(f'\nBayes-Stein Estimate of Mean Return:\n{mu_bs.T}')
print(f'\nBayes-Stein Estimate of Covariance of Return:\n{sigma_bs}')
```

We get the following Bayes-Stein estimates for the mean return on each ETF:

IVV	IEUR	AIA	IEMG
0.0020	0.0017	0.0018	0.0016

We get the following Bayes-Stein estimate of the covariance matrix:

	IVV	IEUR	AIA	IEMG
IVV	0.000536	0.000518	0.000479	0.000486
IEUR	0.000518	0.000746	0.000621	0.000635
AIA	0.000479	0.000621	0.000910	0.000786
IEMG	0.000486	0.000635	0.000786	0.000771

Q4:

Split the returns data in 1 above into two contiguous period of equal length. Use the data in first period to calculate the returns on the global minimum variance (GMVP) and tangency portfolios (TP). Using these weights, generate a time-series of portfolio returns for both GMVP and TP using the return data in the second period. From the time-series of portfolio returns, calculate the Sharpe ratios for the two portfolios and compare their performance.

First we divide our returns data into to contiguous periods of equal* length (*there is an odd number of observations in dataset so we set the number of observations used to fit the model to round up to the nearest integer the number of observations being used to calculate our weights). Based on the first period of returns we calculate the following weights for the GMVP & TP:

	IVV	IEUR	AIA	IEMG
GMVP	0.9696	0.2751	-0.0517	-0.1929
TP	1.6063	-0.3489	1.3608	-1.6183

Our functions for calculating the weights of the GMVP & TP are outlined below.

```
def minvar_weights(returns):

    K = returns.shape[1]
    sigma = returns.cov()

    w_mv = np.divide(np.linalg.inv(sigma) @ np.ones((K,1)), np.ones((1,K)) @ np.linalg.inv(sigma) @ np.ones((K,1)))
    return w_mv

def tangency_weights(returns):

    K = returns.shape[1]
    mu = np.array(returns.mean()).reshape((K,1))
    sigma = returns.cov()

    w_tp = np.divide(np.linalg.inv(sigma) @ mu, np.ones((1,K)) @ np.linalg.inv(sigma) @ mu)
    return w_tp

n = df_hist.shape[0]//2+1
df_train = df_hist.iloc[:n, :]
df_test = df_hist.iloc[n:, :]

w_mv = minvar_weights(df_train)
w_tp = tangency_weights(df_train)
print(f'\Minimum variance portfolio weights\n{w_mv}')
print(f'\nTangency portfolio weights\n{w_tp}')
```

Using these weights, we calculate a time-series of portfolio returns for the second period, storing these in the dataframe 'df_portfolio'. We then use these values to calculate the mean & standard deviation of returns, and then finally come to the Sharpe ratio of each portfolio.

Portfolio	Sharpe Ratio
GMVP	0.0951
TP	0.1100

```
df_portfolio = pd.DataFrame()
df_portfolio['GMVP'] = df_test @ w_mv
df_portfolio['TP'] = df_test @ w_tp

mu_portfolio = df_portfolio.mean()
sd_portfolio = df_portfolio.std()
sharpe_portfolio = round(mu_portfolio/sd_portfolio, 4)
print(f'\nSharpe Ratio of Portfolio:\n{sharpe_portfolio.to_string()}')
```

Q5:

Write a Python function to test differences in Sharpe ratios of two portfolios. Implement the "all-time classic" Jobson and Korkie (1981) test for the equality of the Sharpe ratios (the test statistic is defined in footnote 20, page 271 of the Jorion (1985) article which is attached). There was an error in the original test which has since been corrected by Memmel (2003) also attached. Test your function using the time-series of portfolio returns generated for the GMVP and TP.

We create a function to perform the Jobson-Korkie test given a dataframe containing the returns on two portfolios. We set the default value for the risk-free rate to 0, and the acceptance threshold for our null hypothesis at 5%.

We calculate the mean excess return & covariance matrix of retrurns between the two portfolios, which are then used to calculate the asymptotic variance $\hat{\theta}$, and the transformed difference in sample Sharpe ratios c_{JK} (as per the "all-time classic" Jobson & Korkie test).

These are then used to calculate our test-statisic, $z_{JK} \sim \mathcal{N}(0,1)$. Comparing this to the standard normal distribution, we can then determine the probability of seeing a test-statistic at least as extreme under the null-hypothesis.

```
def jobson_korkie_test(portfolio_returns, risk_free=0, acceptance_threshold=0.05):

    T = len(portfolio_returns)
    mu_a, mu_b = portfolio_returns.mean()-risk_free
    cov_ab = portfolio_returns.cov()

    sigma_a, sigma_b = np.diag(cov_ab)
    sigma_ab = cov_ab.iloc[1,0]

    theta = 1/T*(2*sigma_a*sigma_b-2*np.sqrt(sigma_a*sigma_b)*sigma_ab + 1/2*mu_a**2*sigma_b +
                 1/2*mu_b**2*sigma_a - mu_a*mu_b/(np.sqrt(sigma_a*sigma_b))*sigma_ab**2)
    c_jk = np.sqrt(sigma_b)*mu_a-np.sqrt(sigma_a)*mu_b

    jk_stat = c_jk/np.sqrt(theta)
    # Take negative of absolute value so we get consistent p-value results regardless of ordering of portfolio inputs
    jk_pval = norm.cdf(-abs(jk_stat))
    jk_accept_null = jk_pval > acceptance_threshold/2 # Reject if score in the 5% most extreme results (Two-way test)
    return jk_stat, jk_pval, jk_accept_null

jk_stat, jk_pval, jk_reject_null = jobson_korkie_test(df_portfolio)
print(f'\nTest Statistic: {jk_stat:.4f}')
print(f'Test P-Value: {jk_pval:.4f}')
print(f'Reject Null Hypothesis: {jk_reject_null}')
```

Our function outputs the following test-statistic, p-value, resulting in us failing to reject the null hypothesis that the Sharpe ratios are equal:

Test-statistic	P-Value	Reject Null Hypothesis
-0.3434	0.3657	False