UNIVERSITY OF PRETORIA

DEPARTMENT OF MATHEMATICS AND APPLIED MATHEMATICS

WTW 801 - Big Data : Financial Engineering & Financial Mathematics

External Examiners: Prof. T. Verster (North-West University, Potchestroom)
Internal Examiners: Dr R Kufakunesu and Dr G van Zyl.(University of Pretoria)

Surname:	Initials:
Student number:	
Telephone number:	

GRADES:

TOTAL

Conditions

(1) The deadline for submission is 4 December 2021.

(2) After finishing the project you should upload your .pdf file via the provided ClickUp link.

(3) On the cover page, you should indicate your student details.

(4) Answer all questions.

(5) All sources (including the prescribed texts for this module) which you consult, should be clearly referenced in your project- this should include references to the internet.

(6) Use Python, Matlab, R or any other software. You need to explain your steps and you may put simplifying assumptions where necessary. You may submit your project in a typed version (using for instance Latex or any other word processing software).

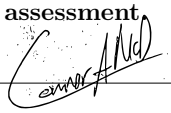
(7) Attach the signed plagiarism form on your project.

(9) Your final report should be a PDF document of less than 20 pages, with a reference list.

(10) Please also remember to add the pseudo/code that you used.

Plagiarism Pledge

The University of Pretoria commits itself to produce academic work of integrity. I affirm that I am aware of and have read the Rules and Policies of the University, more specifically the Disciplinary Procedure and the Tests and Examinations Rules, which prohibit any unethical, dishonest or improper conduct during tests, assignments, examinations and/or any other forms of assessment. I am aware that no student or any other person may assist or attempt to assist another student, or obtain help, or attempt to obtain help from another student or any other person during tests, assessments, assignments, examinations and/or any other forms of assessment.

Signature: 

1 Data Overview

For this question the data consisted of 20 US-based companies that are listed on the OTC Bulletin Board Market. The data was provided by Yahoo Finance via the yfinance python library. The observation period was 2017-01-01 to 2020-12-31 and used a 1-day interval. The specific assets and their respective market sizes can be found in table 1 below:

Table 1: Assets

Ticker	**Name**	**Sector**	**Market Size (USD)**
AAPL	Apple	Technology	2.57T
AMZN	Amazon	E-commerce	1.78T
COST	Costco	Retail	241.29B
ENPH	Enphase Energy	Energy	33.38B
ETSY	Etsy	E-commerce	36.88B
FB	Facebook	Technology	926.66B
FL	Foot Locker	Consumer Discretionary	5.06B
GOOG	Google	Technology	1.89T
LAC	Lithium Americas Corp	Energy	4.29B
MRTX	Mirati Therapeutics	Pharmaceutical	7.55B
NFLX	Netflix	Entertainment	294.85B
NKE	Nike	Consumer Discretionary	265.94B
NVDA	Nvida	Technology	787.58B
PLUG	Plug Power	Energy	23.20B
RILY	B. Riley Financial	Financial Services	2.15B
SEDG	SolarEdge Technologies	Energy	17.93B
SHOP	Shopify	E-commerce	197.99B
TJX	The TJX Companies	Retail	83.41B
TSLA	Tesla	Technology	1.09T
WMT	Walmart	Retail	404.05B

1.a Optimal Portfolio Construction With PCA

1.a.i Data Cleanliness

The data was remarkably clean, and contained no missing or zero values. However, there were a number of duplicates in each column ranging from 8 duplicates to 590 for the smaller companies. However, this is fine since these duplicates are price duplicates and it is possible for a stock to have the same price at different periods in time. Furthermore, the dataset was diverse in terms of what sector each stock was from with a total of 8 different sectors. Lastly, the stocks chosen were some of the fastest growing and most volatile stocks over the last 5 years and had an average annual return around **60%**, thus it is expected to see an optimal portfolio that achieves a higher return than 60% but also has high volatility.

1.a.ii Principal Component Analysis

To perform a principal component analysis on the data, the data was first converted into daily returns for all 20 stocks. The first 5 rows and stocks can be seen in the sample below 2:

Table 2: Daily Returns (Sample)

Date	**AAPL**	**AMZN**	**COST**	**ENPH**	$\cdots$
2017-01-04	-0.001119	0.004657	0.000188	0.095238	$\cdots$
2017-01-05	0.005085	0.030732	0.019717	-0.026087	$\cdots$
2017-01-06	0.011148	0.019912	-0.000491	-0.008929	$\cdots$
2017-01-09	0.009159	0.001168	-0.011423	0.000000	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

The mean daily return of each stock was then calculated with equation 1. This yielded the 20-component mean vector $\vec{\mu}$.

$$\vec{\mu} = [0.00177402, 0.00165138, 0.00105088, 0.00655442, \cdots]$$

$$\mu_i = \frac{\sum_{i=1}^{N} \boldsymbol{x}_i}{N} \tag{1}$$

The mean vector was then subtracted from the daily rates of return matrix to yield the normalised rates of return shown below in table 3.

Table 3: Normalised Daily Returns (Sample)

Date	**AAPL**	**AMZN**	**COST**	**ENPH**	$\cdots$
2017-01-04	-0.002893	0.003006	-0.000863	0.088684	$\cdots$
2017-01-05	0.003311	0.029081	0.018666	-0.032641	$\cdots$
2017-01-06	0.009374	0.018260	-0.001542	-0.015483	$\cdots$
2017-01-09	0.007385	-0.000483	-0.012474	-0.006554	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

A 20x20 covariance matrix was then calculated using the full version of the normalised data shown in table 3. A sample of this covariance matrix (Σ) is shown below.

$$\boldsymbol{\Sigma} = \begin{bmatrix} 0.000398 & 0.000246 & 0.000141 & 0.000353 & \cdots \\ 0.000246 & 0.000372 & 0.000121 & 0.000279 & \cdots \\ 0.000141 & 0.000121 & 0.000195 & 0.000145 & \cdots \\ 0.000353 & 0.000279 & 0.000145 & 0.003018 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

The principle values were calculated by solving for $\lambda_i \epsilon \{\lambda_1, \cdots, \lambda_{20}\}$ in the following:

$$\left|\boldsymbol{\Sigma} - \lambda \boldsymbol{I}\right| = 0$$

$$\begin{vmatrix} 0.000398 - \lambda_1 & 0.000246 & 0.000141 & 0.000353 & \cdots \\ 0.000246 & 0.000372 - \lambda_2 & 0.000121 & 0.000279 & \cdots \\ 0.000141 & 0.000121 & 0.000195 - \lambda_3 & 0.000145 & \cdots \\ 0.000353 & 0.000279 & 0.000145 & 0.003018 - \lambda_4 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{vmatrix} = 0$$

By plotting the squares of the principle values, the scree plot below is generated and shows that **90.66%** of the information can be explained by the first 10 components. These 10 components were subsequently used for the portfolio optimisation.

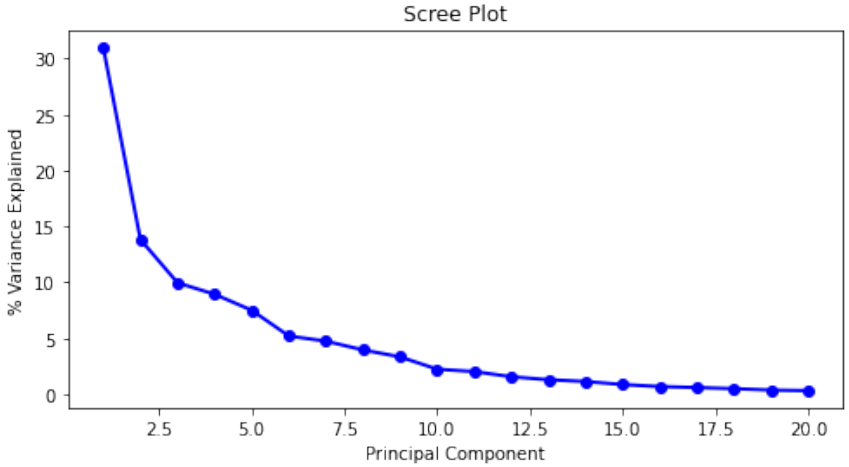


Figure 1

1.a.iii Economic Insights and Relationships

Let $(\boldsymbol{\Sigma} - \lambda_i \mathbf{I}) = \mathbf{B}$ and solve $\mathbf{B}\boldsymbol{v_k} = \bar{0}$, where $\boldsymbol{v_k}$ is the factor loading of the respective principle value (λ_i) and $\boldsymbol{p_k}$ is the unit vector of $\boldsymbol{v_k}$. The proportion of each stock in each factor loading, corresponds to how much that stock affects that factor. The stocks that affected each factor the most were then identified and the results showed that the model was heavily influenced by the energy and e-commerce sectors as 7/10 factors where influenced most by stocks in these two sectors. This is expected as many countries have recently signed global reform pledges to reduce carbon emissions.

Table 4: Most Influential Stocks

F1	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**
ENPH	MRTX	ENPH	PLUG	LAC	FL	TSLA	SEDG	ETSY	SHOP

1.a.iv Factor Portfolio

To generate the optimal factor portfolio with an added volatility constraint the following formula was obtained in the textbook (Cornuejols & Tütüncü 2006) and used below

$$\begin{aligned} \text{maximize} \quad & \mu^T \mathbf{x} \\ \text{subject to} \quad & \mathbf{x}^T \Sigma \mathbf{x} <= \sigma^2, \\ & A\mathbf{x} = b, \\ & C\mathbf{x} >= d \end{aligned} \tag{2}$$

Where:

- μ represents mean returns of each stock
- $\mathbf{x}$ represents the weighting of each factor in the factor portfolio
- Σ represents the covariance matrix of the stock selection
- A and b ensure that the allocation does not exceed 100%
- C and d ensure that no stock receives a negative allocation.

The minimum volatility achievable by this set of factors was 17.7%. Therefore, two optimal portfolios were constructed with added volatility constraints of 20% and 30%, respectively. The weightings of each factor in the two portfolios is shown below:

Table 5: Portfolio Weightings

Volatility	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**
20%	0.052	0.088	0.097	0.099	0.103	0.109	0.110	0.112	0.114	0.117
30%	0.177	0.119	0.104	0.100	0.095	0.086	0.084	0.081	0.079	0.074

1.a.v Investment Problem

To determine which of these portfolios had the highest return, the portfolios had to be converted into stock portfolios instead of factor portfolios. To do this, the matrix of principle components ($\mathbf{p}$) also known as factor loadings was multiplied by the factor weightings ($\mathbf{x}$) calculated above. This yielded the stock weights ($\mathbf{w}$), Example calculations using equation 3 will be shown below.

$$\mathbf{p} \times \mathbf{x} = \mathbf{w} \tag{3}$$

Example Calculation:

$$\boldsymbol{p} = \begin{bmatrix} 0.05 & 0.1 & 0.2 \\ 0.15 & 0.3 & 0.2 \\ 0.45 & 0.4 & 0.3 \\ 0.2 & 0.1 & 0.4 \\ 0.15 & 0.1 & 0.1 \end{bmatrix}; \qquad \boldsymbol{x} = \begin{bmatrix} 0.2 \\ 0.5 \\ 0.3 \end{bmatrix}$$

$$\begin{bmatrix} 0.05 & 0.1 & 0.2 \\ 0.15 & 0.3 & 0.2 \\ 0.45 & 0.4 & 0.3 \\ 0.2 & 0.1 & 0.4 \\ 0.15 & 0.1 & 0.1 \end{bmatrix} \times \begin{bmatrix} 0.2 \\ 0.5 \\ 0.3 \end{bmatrix} = \boldsymbol{w} = \begin{bmatrix} 0.12 & 0.24 & 0.38 & 0.12 & 0.14 \end{bmatrix}$$

Using this methodology, the optimal portfolio for all 20 stocks was constructed for 20% and 30% volatility, the results are shown in the tables below.

Table 6: Optimal Portfolio for **20%** Volatility

Stock	Weight
AAPL	0.024464
AMZN	0.025885
COST	0.014929
ENPH	0.074008
ETSY	0.089740
FB	0.031266
FL	0.081598
GOOG	0.026459
LAC	0.076572
MRTX	0.053078
NFLX	0.033897
NKE	0.022715
NVDA	0.039334
PLUG	0.063282
RILY	0.048864
SEDG	0.083589
SHOP	0.077248
TJX	0.026878
TSLA	0.091315
WMT	0.014881
ANNUAL RETURN	**73.68%**

Table 7: Optimal Portfolio for **30%** Volatility

	Weight
AAPL	0.025795
AMZN	0.027175
COST	0.015063
ENPH	0.084591
ETSY	0.078402
FB	0.030936
FL	0.068454
GOOG	0.026749
LAC	0.080429
MRTX	0.075454
NFLX	0.033929
NKE	0.023289
NVDA	0.041302
PLUG	0.077675
RILY	0.041090
SEDG	0.078183
SHOP	0.068243
TJX	0.026149
TSLA	0.082931
WMT	0.014160
ANNUAL RETURN	**77.0%**

Therefore, it is clear that the portfolio with 30% volatility achieved a higher annual return of 77% for the period 2017-01-01 to 2020-12-31, this would will be the chosen portfolio for the next question.

1.b Portfolio Performance

Table 8 shows the performance of the optimal portfolios vs the market benchmark for the period 2021-01-01 to 2021-06-30. It was observed that both optimal portfolios beat the market benchmark. However, both optimal portfolios had significantly higher variance than the market benchmark. Furthermore, it was observed that the 20% volatility portfolio performed better over this time period, which indicates that the 30% portfolio may be more suited to long term investing. In general the portfolios performed worse than the portfolio constructed for 2017-2020, which achieved annual returns of 73% and 77% for 20% and 30% volatility, respectively. This is likely because many stock had a bearish period between April and June of 2021, and thus the portfolio was not given enough time to recover in this time frame.

Table 8: Performance for 2021-01-01 to 2021-06-30

Portfolio	**Return for period**	**Annual Return**	**Variance**
Optimal (20% volatility)	21.61%	43.92%	6.96%
Optimal (30% volatility)	20.21%	41.07%	7.36%
Cap-weighted	16.98%	34.5%	2.89%

1.c Portfolio Performance During Covid

Repeating the principal component analysisis and portfolio optimisation for the period: 2020-01-01 to 2021-10-31 (COVID-19) the following results and differences were obtained.

Table 9: Daily Returns (Sample)

Date	AAPL	AMZN	COST	ENPH	⋯
2020-01-02	0.022816	0.027151	-0.008268	0.122847	⋯
2020-01-03	-0.009722	-0.012139	0.000824	-0.001704	⋯
2020-01-06	0.007968	0.014886	0.000274	0.012632	⋯
2020-01-07	-0.004703	0.002092	-0.001576	0.011126	⋯
⋮	⋮	⋮	⋮	⋮	⋱

The mean daily return of each stock was then calculated with equation 1. This yielded the 20-component mean vector $\vec{\mu}$.

$$\vec{\mu} = [0.00186501, 0.0015145, 0.00132115, 0.00628798, \cdots]$$

The mean vector was then subtracted from the daily rates of return matrix to yield the normalised rates of return shown below in table 10.

Table 10: Normalised Daily Returns (Sample)

Date	AAPL	AMZN	COST	ENPH	⋯
2020-01-02	0.020951	0.025636	-0.009589	0.116559	⋯
2020-01-03	-0.011587	-0.013654	-0.000498	-0.007992	⋯
2020-01-06	0.006103	0.013371	-0.001047	0.006344	⋯
2020-01-07	-0.006568	0.000577	-0.002897	0.004838	⋯
⋮	⋮	⋮	⋮	⋮	⋱

A 20x20 covariance matrix was then calculated using the full version of the normalised data shown in table 10. A sample of this covariance matrix (Σ) is shown below.

$$\mathbf{\Sigma} = \begin{bmatrix} 0.000584 & 0.000337 & 0.000239 & 0.000579 & \cdots \\ 0.000337 & 0.000424 & 0.000173 & 0.000391 & \cdots \\ 0.000239 & 0.000173 & 0.000236 & 0.000248 & \cdots \\ 0.000579 & 0.000391 & 0.000248 & 0.003182 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

The principle values were calculated by solving for $\lambda_i \epsilon \{\lambda_1, \cdots, \lambda_{20}\}$ in the following:

$$\left|\boldsymbol{\Sigma} - \lambda \boldsymbol{I}\right| = 0$$

$$\begin{vmatrix} 0.000584 - \lambda_1 & 0.000337 & 0.000239 & 0.000579 & \cdots \\ 0.000337 & 0.000424 - \lambda_2 & 0.000173 & 0.000391 & \cdots \\ 0.000239 & 0.000173 & 0.000236 - \lambda_3 & 0.000248 & \cdots \\ 0.000579 & 0.000391 & 0.000248 & 0.003182 - \lambda_4 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{vmatrix} = 0$$

By plotting the squares of the principle values, the scree plot below is generated and shows that **91.56%** of the information can be explained by the first 10 components. These 10 components were used for all subsequwnt calculations

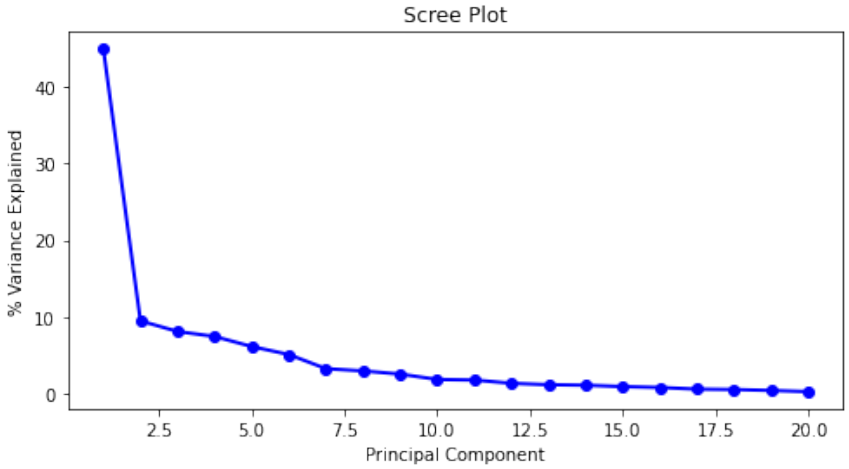


Figure 2

Let $(\boldsymbol{\Sigma} - \lambda_i \mathbf{I}) = \mathbf{B}$ and solve $\mathbf{B}\boldsymbol{v_k} = \bar{0}$, where $\boldsymbol{v_k}$ is the factor loading of the respective principle value (λ_i) and $\boldsymbol{p_k}$ is the unit vector of $\boldsymbol{v_k}$. By solving for the eigenvector matrix $(\boldsymbol{p})$, the most influential stocks of each factor was determined. The model was still largely influenced by the energy sector, however during this period it was also more influenced by the pharmaceutical industry, which is expected as many pharmaceutical companies saw their share prices rise during the pandemic.

Table 11: Most Influential Stocks

F1	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**
PLUG	LAC	ENPH	PLUG	FL	TSLA	ETSY	MRTX	MRTX	SEDG

During this period the selection of stocks was still relatively volatile with a minimum achievable volatility of **17.04%**.

Therefore, two optimal portfolios were constructed with added volatility constraints of 20% and 30%, respectively. The weightings of each factor in the two portfolios is shown below:

Table 12: Portfolio Weightings

Volatility	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**
20%	0.042	0.098	0.101	0.102	0.105	0.110	0.110	0.111	0.112	0.113
30%	0.139	0.100	0.099	0.098	0.097	0.095	0.093	0.093	0.093	0.092

When looking at the portfolio optimised for 30% volatility, the weightings of factors were observed to be more evenly distributed than for the period beginning in 2017-01-01. However, with 20% volatility it appeared that the factor weightings were less evenly distributed. One potential reason for this is that the market experienced higher levels of volatility during this period and thus all factors inherently had more volatility. This causes the model to weight lower volatility factors higher when optimising for lower volatility such as 20%. But the model will be able to weight the factors more evenly for higher volatility such as 30%, if the the majority of stocks were experiencing volatile periods, which they were.

References

Cornuejols, G. & Tütüncü, R. (2006), *Optimization methods in finance*, Vol. 5, Cambridge University Press.

Code For Question 1

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import yfinance as yf
import scipy
import math
import random
import cvxopt
from cvxopt import matrix, solvers
import pypfopt
from scipy.optimize import minimize
\%matplotlib inline

tickers_list = ['TSLA','LAC',
                'AMZN', 'NFLX',
                'WMT', 'COST',
               'FL', 'RILY',
               'ENPH', 'SHOP',
               'PLUG', 'SEDG',
               'ETSY', 'MRTX',
               'TJX', 'NKE',
               'AAPL', 'FB',
               'GOOG', 'NVDA']

df = yf.download(tickers_list,'2017-01-01', '2020-12-31')['Adj Close']

'''
This cell computes the number of missing values in the datframe
'''
stocks = []
nans = []
for i in df.columns:
    stocks.append(i)
    count_nan = df[str(i)].isna().sum()
    nans.append(count_nan)

nan_df = pd.DataFrame(stocks)
nan_df['nans'] = nans
nan_df.columns = ['stock', 'nan values']

#calculate the daily returns of each stock
daily_returns = df.pct_change()
daily_returns = daily_returns.iloc[1:,:]
```

```
44
45 #get average daily returns
46 mean_daily_returns = daily_returns.values.mean(axis=0)
47
48 #Normalise daily returns and calculate covariance matrix, eigen values and
   ↪  eigen vectors
49 Y = daily_returns.values - mean_daily_returns
50 cov = np.cov(Y.T)
51 eigen_values, eigen_vectors = np.linalg.eig(cov)
52
53 #Scree plot
54 x = eigen_values
55 w = []
56 z = []
57 cnt = 1
58 for i in x:
59     w.append(100*(i/sum(x)))
60     z.append(cnt)
61     cnt+=1
62
63 w = sorted(w, key=float, reverse=True)
64 plt.rcParams["figure.figsize"] = (8, 4)
65 plt.plot(z, w,'o-', linewidth=2, color='blue')
66 plt.title('Scree Plot')
67 plt.xlabel('Principal Component')
68 plt.ylabel('% Variance Explained')
69 plt.show()
70
71 x=0
72 for i in range(0,10):
73     x += w[i]
74
75 print(x)
76
77
78 #calculate factor loadings
79 factor_loading = pd.DataFrame(eigen_vectors)
80 factor_loading.columns = ['FL1', 'FL2', 'FL3', 'FL4', 'FL5', 'FL6','FL7',
   ↪  'FL8', 'FL9', 'FL10', 'FL11', 'FL12',
81                   'FL13', 'FL14', 'FL15', 'FL16','FL17', 'FL18', 'FL19',
                     ↪  'FL20']
82 factor_loading.index = daily_returns.columns
83
84
```

```
#Determine highest contributor to each factor
m = []
for i in factor_loading.columns:
    a = min((factor_loading[i]))
    b = max(factor_loading[i])

    if abs(a) > abs(b):
        y =factor_loading[i].to_list()
        y = y.index(a)
        m.append(daily_returns.columns[y])
    else:
        y =factor_loading[i].to_list()
        y = y.index(b)
        m.append(daily_returns.columns[y])

print(m)

#determine the factors to be used
factors = np.matmul(Y , eigen_vectors)
factors = pd.DataFrame(factors)
factors = factors.iloc[:,0:10]
factors.columns = ['F1', 'F2', 'F3', 'F4', 'F5', 'F6','F7', 'F8', 'F9',
↪  'F10']
factors

#Optimisation problem
volatility_constraint = 0.30

def solve_for_max_returns(p):
    p = np.array(p)
    R = np.sum(annual_factor_returns*p)
    return -1*R

def weights_check(p):
    return np.sum(p)-1

def volatility(p):
    p = np.array(p)
    V = np.sqrt(np.dot(p.T,np.dot(cov,p)))
    return V

#inital weights, bounds and constraints
p0 = [0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1]
bounds = ((0,1),(0,1),(0,1),(0,1),(0,1),(0,1),(0,1),(0,1),(0,1),(0,1))
```

```
constraints = ({'type':'eq', 'fun':weights_check},
               {'type':'eq', 'fun': lambda p: volatility(p) -
                ↪  volatility_constraint})
#solve for the optimal factor weightings
p_opt = minimize(solve_for_max_returns, p0, method = 'SLSQP', bounds =
 ↪  bounds, constraints=constraints)
p_opt_weightings = p_opt.x
p_volatility = volatility(p_opt_weightings)*100

#factor weights to list
factor_weights = []
for i in p_opt.x:
    factor_weights.append(i)


#Determine the optimal portfolio
mean_annual_returns = mean_daily_returns*252

factor_loading = pd.DataFrame(eigen_vectors)
factor_loading = factor_loading.iloc[:,0:10]
factor_loading = abs(factor_loading)
factor_loading = factor_loading/factor_loading.sum(axis=0) #weighting of each
 ↪  stock in each factor

mean_annual_returns = mean_daily_returns*252


portfolio_weights = np.matmul(factor_loading,factor_weights)
portfolio_weights

portfolio_stock_returns = portfolio_weights*mean_annual_returns
portfolio_stock_returns

portfolio_annual_return = round(sum(100*portfolio_stock_returns),2)

print(f"Annual Portfolio Return: {portfolio_annual_return}\%")
print(f"Portfolio Volatility: {np.round(p_volatility,)}\%")
```

Code For Question 2

```
# Get Market Benchmark
tickers_list = ['TSLA','LAC',
                 'AMZN', 'NFLX',
                 'WMT', 'COST',
                'FL', 'RILY',
                'ENPH', 'SHOP',
                'PLUG', 'SEDG',
                'ETSY', 'MRTX',
                'TJX', 'NKE',
                'AAPL', 'FB',
                'GOOG', 'NVDA']

df = yf.download(tickers_list,'2021-01-01', '2021-06-30')['Adj Close']

#optimal portfolio
portfolio_weights

#cap weighted portfolio
cap = [2.57, 1.78, 0.24129, 0.03338, 0.033688, 0.92666, 0.00506, 1.89,
↪  0.00429, 0.00755, 0.29485, 0.26594, 0.78758, 0.02320, 0.00215, 0.01793,
↪  0.19799, 0.08341, 1.09, 0.40405]

cap_weights = []
for i in cap:
    cap_weights.append(i/sum(cap))

#calculate the daily returns of each stock
daily_returns = df.pct_change()
daily_returns = daily_returns.iloc[1:,:]
mean_period_returns = daily_returns.values.mean(axis=0)*len(df.index)
mean_period_returns

portfolio_stock_returns = portfolio_weights*mean_period_returns

cap_stock_returns = cap_weights*mean_period_returns

portfolio_period_return = round(sum(100*portfolio_stock_returns),2)
cap_period_return = round(sum(100*cap_stock_returns),2)

print(f"Period Portfolio Return: {portfolio_period_return}\%")
print(f"Period Market Return: {cap_period_return}\%")

#Calculate Variance
```

```
Y = daily_returns.values - daily_returns.values.mean(axis =0)
cov = np.cov(Y.T)
cap_weights = pd.Series(cap_weights)

opt_variance = round(100*np.dot(portfolio_weights.T,
    np.dot(cov,portfolio_weights))*len(df.index),2)

cap_variance = round(100*np.dot(cap_weights.T,np.dot(cov,
    cap_weights))*len(df.index),2)

print(f"Period Portfolio Variance: {opt_variance}%")
print(f"Period Market Variance: {cap_variance}%")
```

Code For Question 3

```
tickers_list = ['TSLA','LAC',
                 'AMZN', 'NFLX',
                 'WMT', 'COST',
                'FL', 'RILY',
                'ENPH', 'SHOP',
                'PLUG', 'SEDG',
                'ETSY', 'MRTX',
                'TJX', 'NKE',
                'AAPL', 'FB',
                'GOOG', 'NVDA']

df = yf.download(tickers_list,'2020-01-01', '2021-10-31')['Adj Close']

'''
This cell computes the number of missing values in the dateframe
'''
stocks = []
nans = []
for i in df.columns:
    stocks.append(i)
    count_nan = df[str(i)].isna().sum()
    nans.append(count_nan)

nan_df = pd.DataFrame(stocks)
nan_df['nans'] = nans
nan_df.columns = ['stock', 'nan values']

#calculate the daily returns of each stock
daily_returns = df.pct_change()
daily_returns = daily_returns.iloc[1:,:]

#get average daily returns
mean_daily_returns = daily_returns.values.mean(axis=0)

#Normalise daily returns and calculate covariance matrix, eigen values and
↪  eigen vectors
Y = daily_returns.values - mean_daily_returns
cov = np.cov(Y.T)
eigen_values, eigen_vectors = np.linalg.eig(cov)

#Scree plot
x = eigen_values
w = []
```

```
z = []
cnt = 1
for i in x:
    w.append(100*(i/sum(x)))
    z.append(cnt)
    cnt+=1

w = sorted(w, key=float, reverse=True)
plt.rcParams["figure.figsize"] = (8, 4)
plt.plot(z, w,'o-', linewidth=2, color='blue')
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('% Variance Explained')
plt.show()

x=0
for i in range(0,10):
    x += w[i]

print(x)


#calculate factor loadings
factor_loading = pd.DataFrame(eigen_vectors)
factor_loading.columns = ['FL1', 'FL2', 'FL3', 'FL4', 'FL5', 'FL6','FL7', 'FL8', 'FL9', 'FL10', 'FL11', 'FL12',
                  'FL13', 'FL14', 'FL15', 'FL16','FL17', 'FL18', 'FL19', 'FL20']
factor_loading.index = daily_returns.columns


#Determine highest contributor to each factor
m = []
for i in factor_loading.columns:
    a = min((factor_loading[i]))
    b = max(factor_loading[i])

    if abs(a) > abs(b):
        y =factor_loading[i].to_list()
        y = y.index(a)
        m.append(daily_returns.columns[y])
    else:
        y =factor_loading[i].to_list()
        y = y.index(b)
```

```
        m.append(daily_returns.columns[y])

print(m)

#determine the factors to be used
factors = np.matmul(Y , eigen_vectors)
factors = pd.DataFrame(factors)
factors = factors.iloc[:,0:10]
factors.columns = ['F1', 'F2', 'F3', 'F4', 'F5', 'F6','F7', 'F8', 'F9',
↪  'F10']
factors

#Optimisation problem
volatility_constraint = 0.30

def solve_for_max_returns(p):
    p = np.array(p)
    R = np.sum(annual_factor_returns*p)
    return -1*R

def weights_check(p):
    return np.sum(p)-1

def volatility(p):
    p = np.array(p)
    V = np.sqrt(np.dot(p.T,np.dot(cov,p)))
    return V

#inital weights, bounds and constraints
p0 = [0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1]
bounds = ((0,1),(0,1),(0,1),(0,1),(0,1),(0,1),(0,1),(0,1),(0,1),(0,1))
constraints = ({'type':'eq', 'fun':weights_check},
                {'type':'eq', 'fun': lambda p: volatility(p) -
                ↪  volatility_constraint})
#solve for the optimal factor weightings
p_opt = minimize(solve_for_max_returns, p0, method = 'SLSQP', bounds =
↪  bounds, constraints=constraints)
p_opt_weightings = p_opt.x
p_volatility = volatility(p_opt_weightings)*100

#factor weights to list
factor_weights = []
for i in p_opt.x:
    factor_weights.append(i)
```

```
126
127
128 #Determine the optimal portfolio
129 mean_annual_returns = mean_daily_returns*252
130
131 factor_loading = pd.DataFrame(eigen_vectors)
132 factor_loading = factor_loading.iloc[:,0:10]
133 factor_loading = abs(factor_loading)
134 factor_loading = factor_loading/factor_loading.sum(axis=0) #weighting of each
    ↪  stock in each factor
135
136 mean_annual_returns = mean_daily_returns*252
137
138
139 portfolio_weights = np.matmul(factor_loading,factor_weights)
140 portfolio_weights
141
142 portfolio_stock_returns = portfolio_weights*mean_annual_returns
143 portfolio_stock_returns
144
145 portfolio_annual_return = round(sum(100*portfolio_stock_returns),2)
146
147 print(f"Annual Portfolio Return: {portfolio_annual_return}\%")
148 print(f"Portfolio Volatility: {np.round(p_volatility,)}\%")
```