Assignment 1: Solution

We first import the necessary packages

In [29]:
```
import pandas            as pd
import numpy             as np
import scipy             as scp
import matplotlib.pyplot as plt
import os as os
from datetime import date as dd
from scipy import stats
from scipy import optimize
import statsmodels.api as sm
from math import sqrt
# %matplotlib inline
```

We now get the data

In [30]:
```
os.getcwd()
df=pd.read_excel('Assignment_1_MF402_2022.xlsx')
os.getcwd()
```

Out[30]:
```
'C:\\Users\\kaush'
```

Part One

In [31]:
```
df.head()
```

Out[31]:

	DATE	US stock returns	US stock index (total return)	dividend yield (per year)	10-year bond returns	10-year bond index	3-month T-bill return	3-month T-bill index	Month inflati ra
0	19500131	0.017764	100.000000	0.0681	-0.006128	100.000000	0.000913	100.000000	-0.0018
1	19500228	0.009971	100.997067	0.0674	0.002117	100.211700	0.000879	100.087929	0.0018
2	19500331	0.004065	101.407625	0.0674	0.000821	100.293983	0.000976	100.185588	0.0005
3	19500429	0.045113	105.982405	0.0661	0.002998	100.594708	0.000834	100.269185	0.0001
4	19500531	0.039292	110.146628	0.0645	0.003311	100.927750	0.001082	100.377685	0.0021

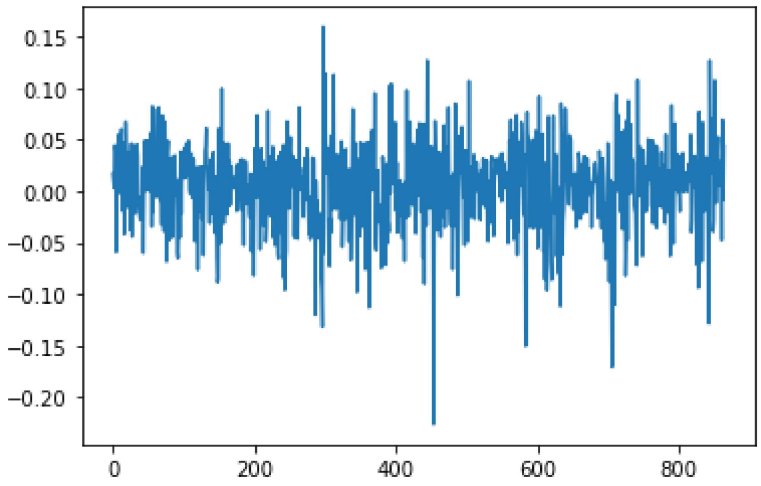
In [32]:
```
date = df.iloc[:,0]
returns = df.iloc[:,1]          # Nominal returns
inflation = df.iloc[:,8]        # Inflation
real_ret = returns - inflation # Real returns

# Fill in rfrate and rf10 from the dataframe
rfrate = df.iloc[:,6]
rf10 = df.iloc[:,4]     # Risk free rate
```

Question 1.1

In [33]:
```
erp = returns - rfrate
plt.plot(erp)
print("excess return: ",erp.mean())
erp_ann= erp*12
print("annualized: ",erp_ann.mean() )
```

```
excess return:  0.003795879909509516
annualized:  0.045550558914114116
```



We now need to run a regression of excess return on a constant. There are multiple ways of doing this. We can define a user created function or use statsmodels package. Here we will use statsmodels package.

In [34]:
```
dim = erp.shape[0]
cons = np.ones((864,), dtype=int)   # Create a vector of ones.
model = sm.OLS(12*erp,cons).fit()
yhat = model.predict()
model.summary()
```

Out[34]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.000
Model:	OLS	**Adj. R-squared:**	0.000
Method:	Least Squares	**F-statistic:**	nan
Date:	Sun, 02 Oct 2022	**Prob (F-statistic):**	nan
Time:	15:28:08	**Log-Likelihood:**	-629.07
No. Observations:	864	**AIC:**	1260.
Df Residuals:	863	**BIC:**	1265.
Df Model:	0		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0456	0.017	2.670	0.008	0.012	0.079

Omnibus:	64.481	**Durbin-Watson:**	1.932
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	137.061
Skew:	-0.456	**Prob(JB):**	1.73e-30
Kurtosis:	4.725	**Cond. No.**	1.00

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

We need to look at the standard errors of the constant. Do they look small or large to you?

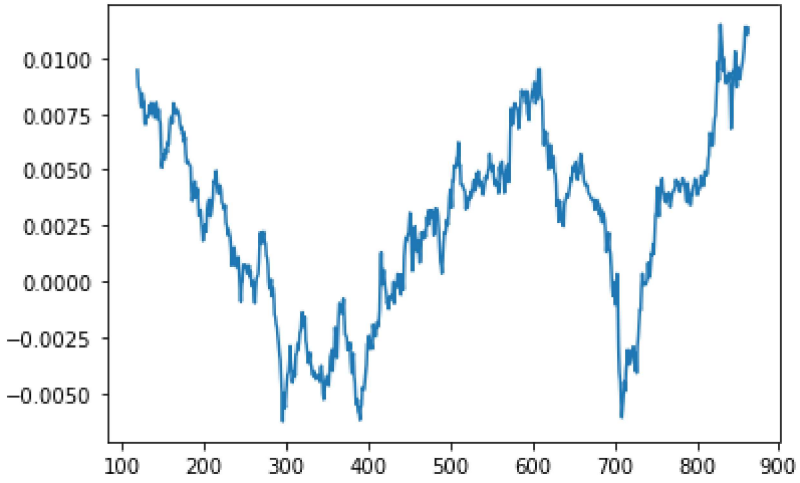
Question 1.2

We are now interested in the moving average of the equity risk premium.

In [35]:
```
ma_erp=erp.rolling(window=120).mean()
```

In [36]:
```
plt.plot(ma_erp)
```

Out[36]:
```
[<matplotlib.lines.Line2D at 0x2281ce91d90>]
```



From the graph above it is clear that the historical estimate of the equity risk premium is stable/not stable.

Question 1.3

We get the long run US GDP growth from the Survey of Professional Forecasters:

https://www.philadelphiafed.org/surveys-and-data/real-time-data-research/spf-q3-2021

2025 GDP Growth rate is xxx

In [37]:
```
import requests
url = "https://www.philadelphiafed.org/surveys-and-data/real-time-data-research/sp
us_gdp_growth = 0.021          # Fill from above
```

From FRED we get the 10 Year TIPS https://fred.stlouisfed.org/series/DFII10#0

For 2021 year end we have: xxx

In [38]:
```
tips_yield = -0.0104
```

In [39]:
```
last_dpratio = df.iloc[dim-1,3]
```

According to the Grodon Growth model the equity risk premium is given by:

In [40]:
```
gordon = last_dpratio*(1+us_gdp_growth) + us_gdp_growth - tips_yield
print("According to the Gordon Growth Model the ERP is %.2f percent" % (gordon*100
```

```
According to the Gordon Growth Model the ERP is 4.46 percent
```

Question 1.4

The novel approach in Slide 9 of the case study gives the following formula:

$$ERP = E/P - RealBondYield$$

In [41]:
```
ep = 4.6/100
erp_nov=ep-tips_yield                                          # Fill in the releva
print(f'according to the novel method, ERP is equal to {erp_nov*100}%')
```

```
according to the novel method, ERP is equal to 5.64%
```

Part Two

Question 2.1

a. We need to calculate the annualized sharpe ratio. We first get the mean return over the full sample and multiply by 12 to annualize, then we get the standard deviation over the full sample and multiply by the square root of 12.

In [42]:
```
erp = returns- rf10
vol_ret = np.std(returns)
annualized_vol = sqrt(12)*vol_ret                # Compute the annnualized volatili

mean_erp = np.mean(erp)
annualized_erp = 12*mean_erp

shratio = annualized_erp/annualized_vol                      # Compute the Sharpe r
print("The annualized sharpe ratio is %.2f" % shratio)
```

```
The annualized sharpe ratio is 0.23
```

b. We now need to calculate the 3% VaR

In [43]:
```
alpha_var = 0.03
returns_sorted = np.sort(returns)

VaR = np.quantile(returns_sorted, alpha_var) # quantile interpolates (linear defau
print("The VaR is %.2f percent" % (VaR*100))
```

```
The VaR is -7.91 percent
```

c. Calculating Expected Shortfall

In [44]:
```
alpha_es = 0.03
es_sorted = [x for x in returns_sorted if x<= -0.0791]
ES = np.mean(es_sorted)           # COmpute the expected shortfall
print(es_sorted)
print("\nThe Expected Shortfall is %.2f percent" % (ES*100))
```

```
[-0.2176304260013051, -0.16942453444905536, -0.14579671089616042, -0.1251193208359
5656, -0.1193347193347194, -0.11386092898697953, -0.1100134269215236, -0.109931224
8752844, -0.10179482667605144, -0.09431419345781267, -0.09229068601254742, -0.0917
7695576721712, -0.09157401989467528, -0.09079145327128291, -0.09048309717728453, -
0.09027865338544949, -0.08599019006744313, -0.0859623816392695, -0.085657348463880
51, -0.08543865891748703, -0.0853489018626632, -0.08411046900964814, -0.0819759162
0389466, -0.08183800100620497, -0.08172338961520131, -0.0800685602350637]

The Expected Shortfall is -10.36 percent
```

d. Calculating Skew

In [45]:
```
sk = scp.stats.skew(returns)
print("The skewness is %.2f" % sk)
```

```
The skewness is -0.43
```

e. Calculating Kurtosis

In [46]:
```
kurt = scp.stats.kurtosis(returns)
print("The kurtosis is %.2f" % kurt)
```

```
The kurtosis is 1.70
```

f. We now need to calculate Maximum Drawdown. We use the python code used in class to have a user written function.

In [47]:
```
def maxdrawdown(x):
    """
    NumPy analog to corresponding MATLAB Financial Toolbox Function
    OUTPUT: start, end, and max drawdown in terms of max percentage drop from a pea
    """
    i = np.argmax(np.maximum.accumulate(x) - x) # end of period
    j = np.argmax(x[:i]) # start of the period
    percent_change = 100*( x[i] - x[j] ) / x[j]
    return j,i,percent_change

def datenum(date):
    """
    Converts yyyymmdd dates to ordinal code
    """
    dn = np.zeros(date.shape)
    for i in range(len(date)):
        dt = str(date[i])
        dn[i] = dd.toordinal(dd(int(dt[0:4]),int(dt[4:6]),int(dt[6:8])))
    return dn
```

In [48]:
```
cumulativeret = np.cumprod(1+returns)

MaxDDStartIndex, MaxDDEndIndex, MaxDD = maxdrawdown(cumulativeret)
begdate = date[0]
enddate = date[len(date)-1]
```

In [49]:
```
print(f'The Maximum Drawdown is {MaxDD:.2f}%')
```

```
The Maximum Drawdown is -52.56%
```

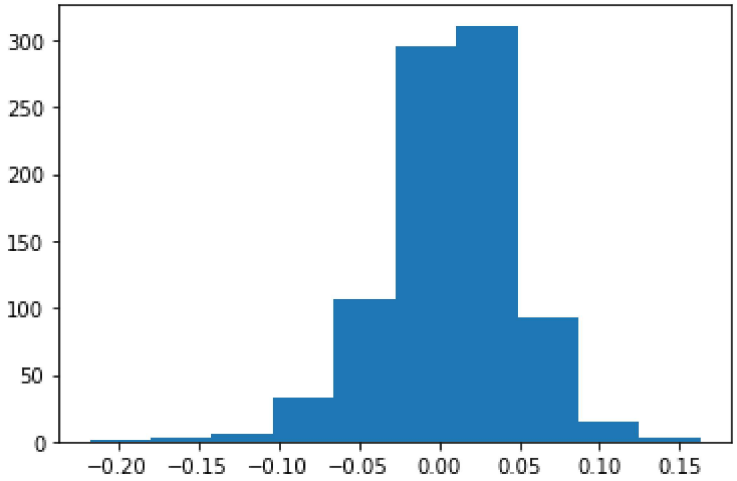
In [50]:
```
print(f'Max VC DrawDown from date {begdate} to date {enddate}')
```

```
Max VC DrawDown from date 19500131 to date 20211231
```

g. Checking normality

In [51]:
```
plt.hist(returns)
```

Out[51]:
```
(array([  1.,   2.,   5.,  32., 107., 295., 311.,  93.,  15.,   3.]),
 array([-0.21763043, -0.17956269, -0.14149496, -0.10342723, -0.0653595 ,
        -0.02729176,  0.01077597,  0.0488437 ,  0.08691143,  0.12497917,
         0.1630469 ]),
 <BarContainer object of 10 artists>)
```



Looking at the histogram, skewness and kurtosis, we conclude xxx.

Question 2.2

We first setup the Sharpe ratio function for the optimization problem.

In [52]:
```
def Shratio(w,mu,Sigma,Rf):
    portfolio_return = np.sum(mu * w) - Rf    # portfolio excess return
    portfolio_vol    = np.sqrt(np.dot(w.T, np.dot(Sigma,w)))  # portfolio volatili
    return portfolio_return / portfolio_vol
```

We now need to set the risk free rate and select the number of assets in our portfolio.

In [55]:
```
Rf = rfrate[len(rfrate)-1]  # risk free rate
n = 2                       # number of assets
tbonds_ret = df.iloc[:,4]   # tbonds
tbills_ret =df.iloc[:,6]    # tbills
```

Combine stock, and bond returns in to a matrix, calculate the mean return of all assets, and the covariance.

In [56]:
```
ret = np.concatenate((np.expand_dims(returns,1), np.expand_dims(tbonds_ret,1)),axi
mu = np.mean(ret, axis = 0)
Sigma = np.cov(ret, rowvar = False)
```

Set the initial weights to be equal

In [57]:
```
w0 = np.ones((n,1))/n
```

Set the constraint condition and solve for the tangency portfolio.

In [58]:
```
cons = ({'type' : 'eq', 'fun': lambda x: x[0] + x[1] - 1}) # portfolio weights sum

# solve for tangency portfolio
sol = scp.optimize.minimize(lambda w: -Shratio(w, mu, Sigma, Rf), x0 = w0, constra
w_uncon = sol.x
val = -sol.fun

print(f'optimal weights are {w_uncon} with Sharpe ratio {val:.2}')
```

```
optimal weights are [0.27653573 0.72346427] with Sharpe ratio 0.28
```

Calculate the mean and variance of the tangency portfolio, and the weight on the portfolio.

In [59]:
```
#### add risk-free asset
gam = 5                                          # gamma: risk aversion coefficient
mu_uncon  = w_uncon.T.dot(mu)                    # mean of tangency portfolio
var_uncon = np.dot(w_uncon.T,np.dot(Sigma,w_uncon))
wT_con    = (mu_uncon-Rf)/ (gam*var_uncon)                                  # wei
rcon      = wT_con*mu_uncon + (1-wT_con)*Rf
```

In [60]:
```
print(f'The mean of the tnagency portfolio is :{mu_uncon:.2}')
print(f'The variance of the tangency portfolio is : {var_uncon:.2}')
print(f'The weight on the risky portfolio is : {wT_con:.2}')
print(f'The return of the tangency portfolio is : {rcon:.2}')
```

```
The mean of the tnagency portfolio is :0.0055
The variance of the tangency portfolio is : 0.00038
The weight on the risky portfolio is : 2.8
The return of the tangency portfolio is : 0.015
```

Question 2.3

In [61]:
```
delta = tbonds_ret - returns
under=sum(delta > 0) / dim
```

In [62]:
```
print(f'a. Stocks underperform bonds approx. %.2f %% of the time.'%under)
```

```
a. Stocks underperform bonds approx. 0.44 % of the time.
```

In [63]:
```
def CumReturnsAndDeltas(df, num_months, dim):

    ret =[]
    delta = []

    for i in range(num_months, dim):
        ret.append(np.array([(df.iloc[i,2] / df.iloc[i-num_months,2]) - 1,
                             ( df.iloc[i,5] / df.iloc[i-num_months,5] ) - 1,
                             ( df.iloc[i,7] / df.iloc[i-num_months,7] ) - 1,
                             ( df.iloc[i,11] / df.iloc[i-num_months,11] ) - 1,
                             ( df.iloc[i,14] / df.iloc[i-num_months,14] ) - 1,
                             ( df.iloc[i,9] / df.iloc[i-num_months,9] ) - 1 ]))
        delta.append(ret[i-num_months][1]- ret[i-num_months][0])

    ret = np.array(ret)
    delta = np.array(delta)
    return ret, delta
```

In [64]:
```
ret_cumulative_1yr, delta_1yr = CumReturnsAndDeltas(df, 12, dim)
delt_1yr=sum(delta_1yr > 0) / delta_1yr.shape[0]

ret_cumulative_5yr, delta_5yr = CumReturnsAndDeltas(df, 60, dim)
delt_5yr=sum(delta_5yr > 0) / delta_5yr.shape[0]

ret_cumulative_10yr, delta_10yr = CumReturnsAndDeltas(df, 120, dim)
delt_10=sum(delta_10yr > 0) / delta_10yr.shape[0]
```

In [65]:
```
print( f'b.i At the 1 year horizon stocks underperformed bonds {delt_1yr*100:.4f}%
```

```
b.i At the 1 year horizon stocks underperformed bonds 39.2019% of the time.
```

In [66]:
```
print(f'b.ii At the 5 year horizon stocks underperformed bonds {delt_5yr*100:.4}% (
```

```
b.ii At the 5 year horizon stocks underperformed bonds 39.93% of the time.
```

In [67]:
```
print(f'b.iii At the 10 year horizon stocks underperformed bonds {delt_10*100:.4}%
```

```
b.iii At the 10 year horizon stocks underperformed bonds 42.2% of the time.
```

PART 3

Try it yourself (Its very similar to above.)

In [68]:
```
def CumReturns(df, num_months, dim):

    ret =[]
    for i in range(num_months, dim):
        ret.append(np.array([(df.iloc[i,2] / df.iloc[i-num_months,2]) - 1,
                             ( df.iloc[i,7] / df.iloc[i-num_months,7] ) - 1,
                             ( df.iloc[i,11] / df.iloc[i-num_months,11] ) - 1,
                             ( df.iloc[i,14] / df.iloc[i-num_months,14] ) - 1,
                             ( df.iloc[i,9] / df.iloc[i-num_months,9] ) - 1 ]))
    ret = np.array(ret)
    return ret
```

3 MONTH

In [69]:
```
threemonth= CumReturns(df, 3, dim)
threemonthdf = pd.DataFrame(threemonth, columns= ['Stocks', '3mth T-Bills', 'Bitco
threemonthdf.corr()
```

Out[69]:

	Stocks	3mth T-Bills	Bitcoin	Gold	Inflation
Stocks	1.000000	-0.080677	0.321422	-0.016667	-0.069724
3mth T-Bills	-0.080677	1.000000	-0.106276	0.154692	0.523126
Bitcoin	0.321422	-0.106276	1.000000	-0.131193	0.090995
Gold	-0.016667	0.154692	-0.131193	1.000000	0.039742
Inflation	-0.069724	0.523126	0.090995	0.039742	1.000000

12 MONTH

In [70]:
```
twelve_month= CumReturns(df, 12, dim)
twelve_monthdf = pd.DataFrame(twelve_month, columns= ['Stocks', '12mth T-Bills', '
twelve_monthdf.corr()
```

Out[70]:

	Stocks	12mth T-Bills	Bitcoin	Gold	Inflation
Stocks	1.000000	-0.100354	0.490674	-0.068439	-0.175760
12mth T-Bills	-0.100354	1.000000	-0.287421	0.386944	0.645586
Bitcoin	0.490674	-0.287421	1.000000	-0.058731	0.194255
Gold	-0.068439	0.386944	-0.058731	1.000000	0.232671
Inflation	-0.175760	0.645586	0.194255	0.232671	1.000000

24 MONTH

In [71]:
```
twentyfour_month= CumReturns(df, 24, dim)
twentyfour_monthdf = pd.DataFrame(twentyfour_month, columns= ['Stocks', '24mth T-B
```

```
twentyfour_monthdf.corr()
```

Out[71]:

	Stocks	24mth T-Bills	Bitcoin	Gold	Inflation
Stocks	1.000000	-0.067178	0.384509	-0.193477	-0.233945
24mth T-Bills	-0.067178	1.000000	-0.476236	0.490034	0.706779
Bitcoin	0.384509	-0.476236	1.000000	-0.175044	0.120077
Gold	-0.193477	0.490034	-0.175044	1.000000	0.422331
Inflation	-0.233945	0.706779	0.120077	0.422331	1.000000

In [ ]: