Youssef Mahmoud 905854027

In [ ]:

```
import pmdarima as pm
import scipy.stats as st
import yfinance as yf
import matplotlib.pyplot as plt
import pandas as pd
import warnings
import numpy as np
import datetime
import io
import seaborn as sns
import itertools as it
import datetime
import matplotlib.lines as mlines
from fredapi import Fred
import statsmodels.formula.api as smf
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
warnings.filterwarnings('ignore')
from fredapi import Fred
```

1

a

In [2]:
```
sectors = pd.read_excel('forwardSector.xlsx', index_col = 0, parse_
sectors.head()
```

Out[2]:

Date	S5MATR Index (R1)	S5ENRS Index (R2)	S4RLST Index (L1)	S5FINL Index (R1)	S5INDU Index (R1)	S5TELS Index (R1)	S5UTIL Index (R1)	S5COND Index (L1)	S5CONS Index (R1)	S5I I
2023-01-31	17.9081	10.7430	32.2286	13.3426	19.1048	15.5487	18.0203	22.8799	20.3698	17.
2023-01-27	17.5719	10.9352	31.7015	13.2034	18.9574	15.6275	18.0010	22.8657	20.1476	17.
2023-01-20	17.1242	10.6727	30.3745	12.8771	18.3096	14.9715	18.0841	21.3794	20.1048	17.
2023-01-13	17.3470	10.5296	29.7772	13.0073	18.8523	14.5053	18.6572	21.3390	20.7692	17.
2023-01-06	16.5958	10.1140	28.1291	12.7110	18.7490	13.9931	18.5873	19.9729	21.1234	17.

In [3]:
```
sectors = sectors.loc['1998-12-21':'2023-03-07',:]
sectors.sort_index(inplace = True)
sectors_health = pd.DataFrame(sectors['S5HLTH Index  (R1)'])
```

In [4]:
```
# pull in prices
sector = 'XLV'
sector = yf.download(sector)[['Adj Close']].copy()
```

```
[*********************100%***********************]  1 of 1 completed
```

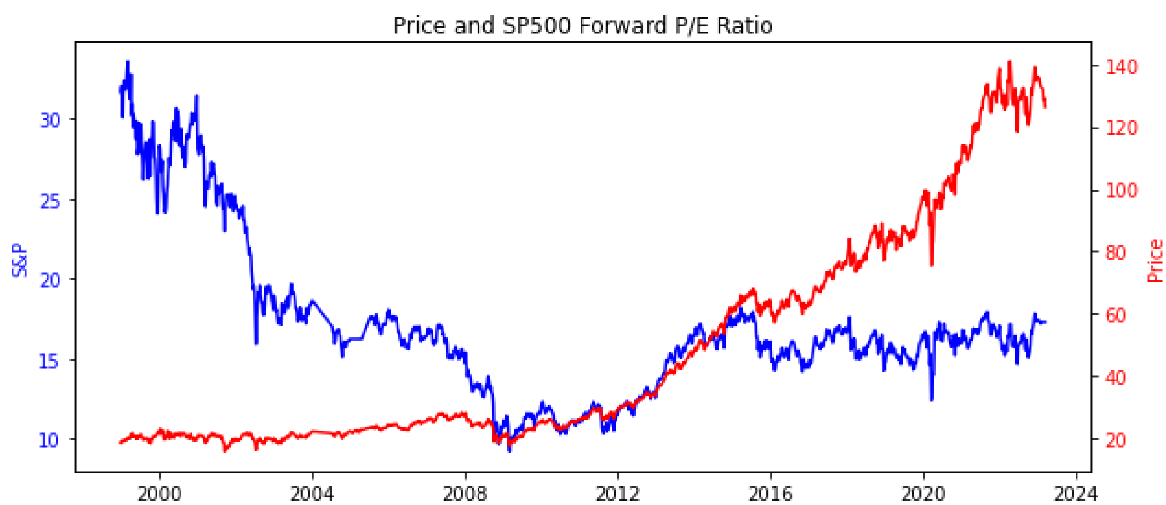
In [5]:
```
data = pd.merge_asof(sector, sectors_health, left_index = True, rig
data = data.resample('W').last()
data['returns'] = np.log(data['Adj Close']).diff()
data.dropna(inplace = True)
data
```

Out[5]:

Date	Adj Close	S5HLTH Index (R1)	returns
1999-01-03 00:00:00-05:00	18.404633	31.6983	0.009662
1999-01-10 00:00:00-05:00	19.101444	32.0664	0.037162
1999-01-17 00:00:00-05:00	18.990839	30.9403	-0.005807
1999-01-24 00:00:00-05:00	18.880238	30.0970	-0.005841
1999-01-31 00:00:00-05:00	19.289473	32.4037	0.021444
...	...	...	...
2023-02-12 00:00:00-05:00	132.490005	17.3003	-0.001508
2023-02-19 00:00:00-05:00	131.990005	17.3003	-0.003781
2023-02-26 00:00:00-05:00	128.509995	17.3003	-0.026720
2023-03-05 00:00:00-05:00	129.169998	17.3003	0.005123
2023-03-12 00:00:00-05:00	126.349998	17.3003	-0.022074

1226 rows × 3 columns

In [6]:
```
fig, ax1 = plt.subplots(figsize = (9,4))

color = 'b'
ax1.set_ylabel('S&P', color = color)
ax1.plot(data['S5HLTH Index  (R1)'], color = color)
ax1.tick_params(axis = 'y', labelcolor = color)

ax2 = ax1.twinx()  # instantiate a second axes that shares the same

color = 'r'
ax2.set_ylabel('Price', color = color)  # we already handled the x-
ax2.plot(data['Adj Close'], color = color)
ax2.tick_params(axis = 'y', labelcolor = color)

plt.title('Price and SP500 Forward P/E Ratio')

fig.tight_layout()  # otherwise the right y-label is slightly clipp
plt.show()
```



b

In [7]:

```
# specify candidate values for hyperparameters
ks = np.linspace(.01, .99, 10)
zs = np.linspace(.01, 3, 10)
ws = np.arange(5, 30, 5)
hs =  1

# build a grid with all possible hyperparameters
grid = np.array(np.meshgrid(ks, zs, ws, hs)).T.reshape(-1,4)
grid
```

Out[7]:

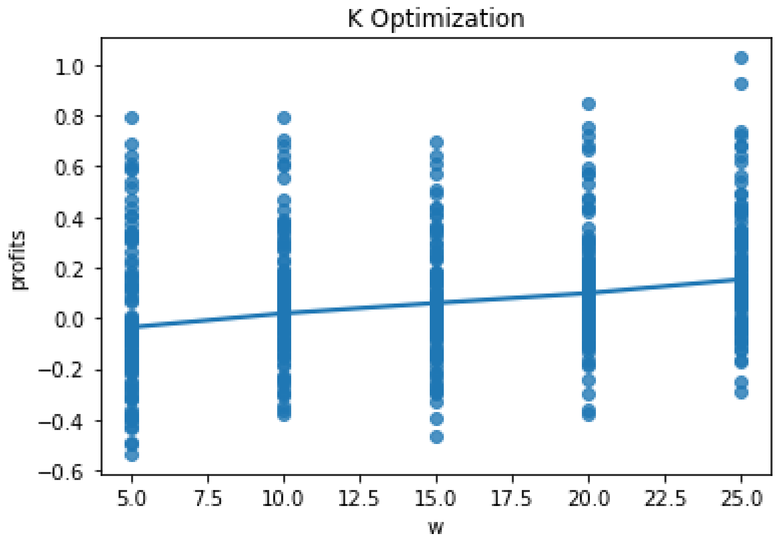
```
array([[1.00000000e-02, 1.00000000e-02, 5.00000000e+00, 1.00000000e+0
0],
       [1.00000000e-02, 3.42222222e-01, 5.00000000e+00, 1.00000000e+0
0],
       [1.00000000e-02, 6.74444444e-01, 5.00000000e+00, 1.00000000e+0
0],
       ...,
       [9.90000000e-01, 2.33555556e+00, 2.50000000e+01, 1.00000000e+0
0],
       [9.90000000e-01, 2.66777778e+00, 2.50000000e+01, 1.00000000e+0
0],
       [9.90000000e-01, 3.00000000e+00, 2.50000000e+01, 1.00000000e+0
0]])
```

In [8]:
```
storage = pd.DataFrame(columns = ['k', 'z', 'w', 'h','profits'])
df_copy = data.copy()

# splitting our data in 80% training and 20% testing
split = 0.8
train_size = int(len(df_copy) * split)
train, test = df_copy[0:train_size], df_copy[train_size:]

for n in range(len(grid)):
    # Each loop we pull out the values for a new set of hyperparamt
    k, z, w, h = grid[n]

    # We fit a filter and CI using the next set of parameters
    train['Filter'] = train['S5HLTH Index  (R1)'].ewm(alpha = k, ad
    train['Filter Error'] = train['S5HLTH Index  (R1)'] - train['Fi
    train['std'] = train['Filter Error'].rolling(int(w)).std()
    train['Upper'] = train['Filter'] + z*train['std']
    train['Lower'] = train['Filter'] - z*train['std']
    train['test'] = np.where(train['Filter Error'].abs()>z*train['s

    # create vectors where we can store information on signals and
    train['test2'] = 0
    train['signal'] = 0

    for j in train.index:
        # if there is a change in the signal, we want to take a pos
        if (train.loc[j, 'test'] == 1) &  (train.shift().loc[j, 'te
            train.loc[j:j+datetime.timedelta(h), 'signal'] = 1
            train.loc[j, 'test2'] = 1
        elif (train.loc[j, 'test'] == -1) &  (train.shift().loc[j,
            train.loc[j:j+datetime.timedelta(h), 'signal'] =  -1
            train.loc[j, 'test2'] = -1

    # calculate metric
    train['cumulative_returns'] = np.exp((train['signal'].shift()*t

    # store the results
    storage = storage.append({'k':k, 'z':z, 'w':w, 'h':h,
                  'profits':train['cumulative_returns'][-1]}, ignore_
```

In [9]:
```
storage = storage.sort_values(by = 'profits').reset_index(drop = Tr
```

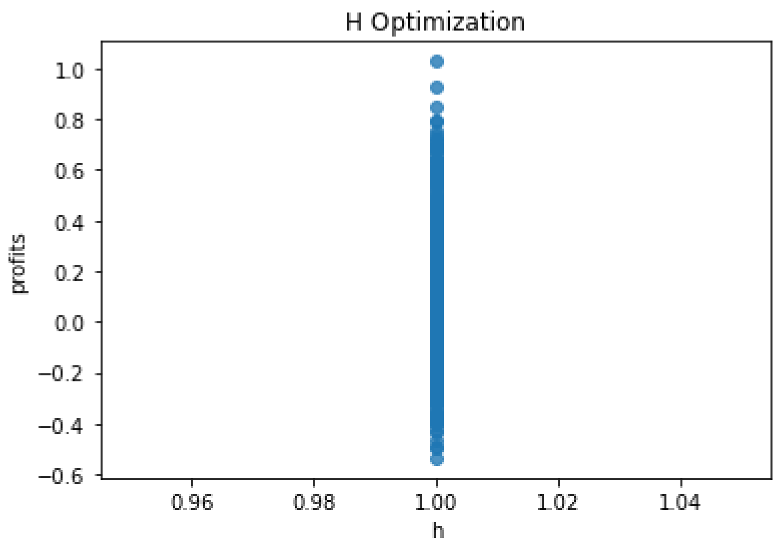
In [10]:

```
sns.regplot(data = storage, x = 'w', y = 'profits', lowess = True)
plt.title('K Optimization')
plt.show()
```



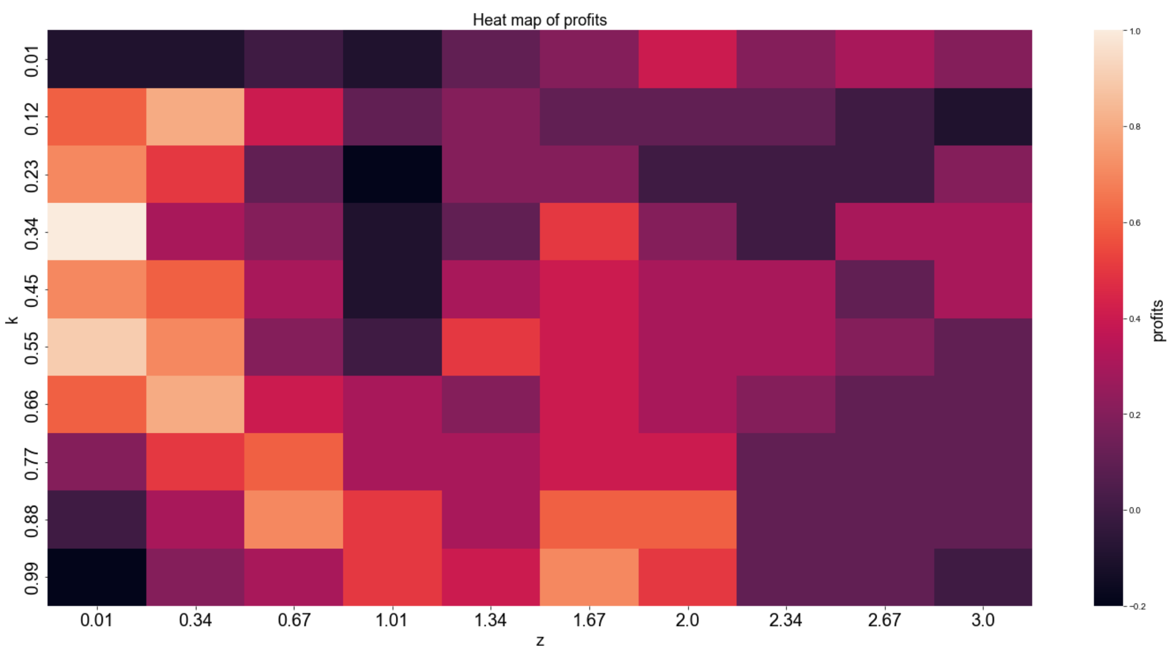
In [11]:

```
sns.regplot(data = storage, x = 'h', y = 'profits', lowess = True)
plt.title('H Optimization')
plt.show()
```

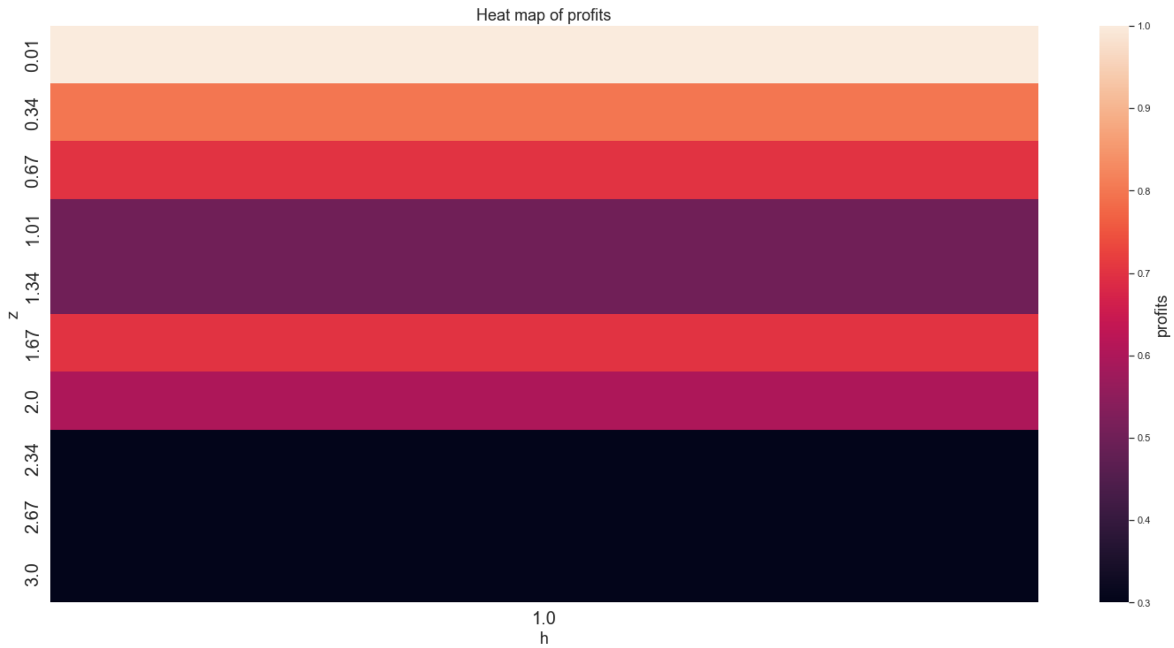


In [12]:
```
def heatmap(x, y, metric, values):

    # specify the columns I will be pulling from the results
    p2p = values[[x, y, metric]]

    # If p > 2, we need to group
    heat = np.round(p2p.groupby([x,y]).max(), 1)
    heat = heat.unstack()[metric]

    # round labels
    heat.index = np.round(heat.index, 2)
    heat.columns = np.round(heat.columns, 2)

    # make plot
    f, ax = plt.subplots(figsize = (25, 12))
    ax = sns.heatmap(heat, fmt = '.1g')
    ax.set_title('Heat map of '+ metric,size = 18)
    ax.tick_params(axis = 'both', which = 'major', labelsize=20)
    ax.set_xlabel(y, size = 18)
    ax.set_ylabel(x, size = 18)
    ax.collections[0].colorbar.set_label(metric, size = 18)
    sns.set(font_scale = 1)
    plt.show()
```

In [13]:
```
heatmap('k', 'z', 'profits', storage)
```



In [14]:
```
heatmap('z', 'h', 'profits', storage)
```



In [15]:
```
storage.iloc[-1]
```

Out[15]:
```
k           0.336667
z               0.01
w               25.0
h                1.0
profits     1.029057
Name: 499, dtype: object
```

In [16]:
```
k, z, w, h, p = storage.iloc[-1]
k, z, w, h, p =  (0.336667, 0.01, 25.0, 1.0, 1.029057)
```

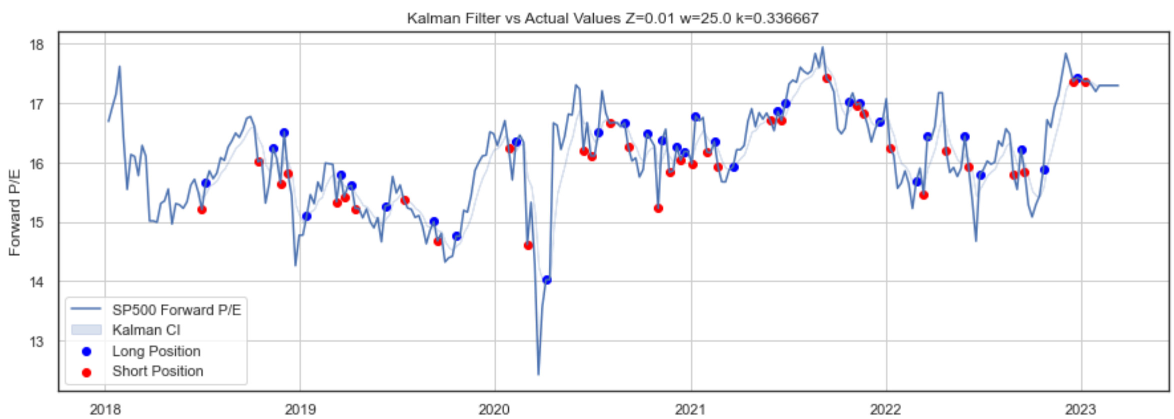
The selection of k, z, w, and h correspond to our heatmap results.

c

In [17]:

```
test = df_copy[int(train_size - w):]
test['Filter'] = test['S5HLTH Index  (R1)'].ewm(alpha = k, adjust =
test['Filter Error'] = test['S5HLTH Index  (R1)'] - test['Filter']
test['std'] = test['Filter Error'].rolling(int(w)).std()
test['Upper'] = test['Filter'] + z*test['std']
test['Lower'] = test['Filter'] - z*test['std']

test['test'] = np.where(test['Filter Error'].abs()>z*test['std'], 1

test['test2'] = 0
test['signal'] = 0
for j in test.index:
    if (test.loc[j, 'test'] == 1) &  (test.shift().loc[j, 'test'] !
        test.loc[j:j+datetime.timedelta(h), 'signal'] = 1
        test.loc[j, 'test2'] = 1
    elif (test.loc[j, 'test'] == -1) &  (test.shift().loc[j, 'test'
        test.loc[j:j+datetime.timedelta(h), 'signal'] =  -1
        test.loc[j, 'test2'] = -1
test['cumulative_returns'] = (np.exp((test['signal'].shift()*test.r
test['strat_returns'] = test['signal'].shift()*test.returns
```

In [22]:
```
sns.set_style('white')
fig, ax = plt.subplots(figsize = (15, 5))
ax.set_title('Kalman Filter vs Actual Values ' + 'Z='+str(z) + ' w=

ax.set_ylabel('Forward P/E')
ax.plot(test['S5HLTH Index  (R1)'])
#ax.plot(df_copy["Filter"])


ax.fill_between(test.index, test.Lower, test.Upper, color='b', alph
ax.scatter(test[test.test2 == 1].index, test[test.test2 == 1]['S5HL
ax.scatter(test[test.test2 == -1].index, test[test.test2 == -1]['S5

#ax.axvline(df_copy.index[-5], color = "red", linestyle = '--')
ax.legend(['SP500 Forward P/E', 'Kalman CI', 'Long Position', 'Shor


#plt.xlim([datetime.date(2000, 1, 1), datetime.date(2004, 1, 1)])
ax.grid()
```



In [23]:
```
test['success'] = ((test[test.test2 != 0]['cumulative_returns'].dif
```

In [25]:

```
fig, ax = plt.subplots(figsize = (15, 5))

plt.title('SP500 Index and Trading Positions'+ ', Z = '+str(z) + ',
plt.ylabel('SP500')
plt.plot(test['Adj Close'])

longsuccess = test[(test['success'] == 1) & (test['test2'] == 1)]
longfail = test[(test['success'] == 0) & (test['test2'] == 1)]
shortsuccess = test[(test['success'] == 1) & (test['test2'] == -1)]
shortfail = test[(test['success'] == 0) & (test['test2'] == -1)]

plt.scatter(longsuccess.index, longsuccess['Adj Close'], color = 'b
plt.scatter(longfail.index, longfail['Adj Close'], color = 'blue',

plt.scatter(shortsuccess.index, shortsuccess['Adj Close'], color =
plt.scatter(shortfail.index, shortfail['Adj Close'], color = 'red',

plt.legend(['Index', 'Long Success', 'Long Fail', 'Short Success',
#plt.xlim([datetime.date(2022, 1, 1), datetime.date(2023, 1, 1)])
plt.grid()
```

In [26]:
```
fig, ax = plt.subplots(figsize = (15, 5))

(test['cumulative_returns']).plot()

plt.title('Price Earnings Strategy:'+ ' Z = '+str(z) + ', w = ' + s

plt.legend(['Equity Curve'])

plt.grid()
#plt.xlim(["1/1/2019", "1/1/2022"])
```

Total rate of return:

In [27]:
```
P = 1000
A = 1000 *((test['cumulative_returns'][-1]/100)+1)
T = len(data)/12
ccror = np.log(A/P)/T
ccror
```

Out[27]: -0.0014489193969168606

Annualized Return

In [28]:
```
annual = (A/P)**(1/T)-1
annual
```

Out[28]: -0.0014478702199929305

Rate of return only over the days we hold a position

In [30]:
```
T = len(test[test['signal'] != 0])/12
(np.log(A/P)/T) *100
```

Out[30]: -2.6914775463940477

Sharpe Ratio

In [32]:
```
test
```

Out[32]:

Date	Adj Close	S5HLTH Index (R1)	returns	Filter	Filter Error	std	Upper	Lo
2018-01-07 00:00:00-05:00	78.252663	16.6916	0.030610	16.691600	0.000000	NaN	NaN	
2018-01-14 00:00:00-05:00	79.693802	16.9304	0.018249	16.771996	0.158404	NaN	NaN	
2018-01-21 00:00:00-05:00	81.180847	17.1492	0.018488	16.898988	0.250212	NaN	NaN	

In [33]:
```
rf_rate = pd.read_csv('1-year-treasury-rate-yield-chart.csv', index
drange = pd.date_range(start = '01/2018', end = '03/17/2023', freq
test.index = drange
test['rf'] = rf_rate
```

In [34]:
```
# add the monthly risk free rate to our data
test['rf'] = rf_rate
test['rf'] = test['rf'].ffill()
```

In [35]:
```
# Subset strategy returns
return_frame = test[['strat_returns']].copy().dropna()

# Subset the monthly rate of raturn for rthe risk free rate
return_frame['rf'] = (test[['rf']].dropna()/100+1)**(1/12)-1

excess_return = return_frame['strat_returns']- return_frame['rf']
```

In [36]:
```
# annualized return method (some use the arithmetic return, which i
anualized_excess = ((excess_return+1).prod()**(12/len(test))-1)*100
```

In [37]:
```
# calculate the anualized standard deviation
excess_ann_std = excess_return.std()*np.sqrt(12)*100
```

In [38]:
```
anualized_excess/excess_ann_std
```

Out[38]: -0.5504062724600376

- Gini Coefficient

In [39]:
```
def GINI_COEFF(returns):
    # get the number of periods -> will allow us to calculate the a
    periods = len(returns)

    # sort values and sum to calculate the Lorenz curve
    LorenzCurve = np.cumsum(returns.sort_values(by = 'strat_returns

    # start from 0:
    LorenzCurve = pd.DataFrame({'strat_returns': [0]}).append(Loren
    Line = LorenzCurve.copy()
    # form the line that encompasses A and B
    Line['strat_returns'] = np.arange(0, 1+1/periods, 1/periods) * 

    # calculate the area of A+B
    UpArea = 0
    for i in range(1, len(returns)):
        UpArea = UpArea + ((Line.iloc[i, :] - LorenzCurve.iloc[i, :
                           + Line.iloc[i-1, :] - LorenzCurve.iloc[

    #calculate the area of A+B+C
    if min(LorenzCurve['strat_returns']) < 0:
        AllArea = ((np.abs(min(LorenzCurve['strat_returns'])) * per
        ((max(LorenzCurve['strat_returns']) * periods)/2))
    else:
        AllArea = ((max(LorenzCurve['strat_returns']) * periods)/2)

    gini = UpArea/AllArea
    return print('Gini Coefficient:' , gini[0])
```

In [42]:
```
returns = test[['strat_returns']][1:]
returns.columns = ['strat_returns']
GINI_COEFF(returns)
```

Gini Coefficient: 0.9402857259735234

2

a

In [43]:
```
semi_annual_r = 0.035/2
n_comp_periods = 2*10

semi_annual_c = semi_annual_r*1000

semi_annual_ytm = (semi_annual_c + (1000-983.125)/n_comp_periods) /
semi_annual_ytm
```

Out[43]: 0.018499842420422314

b

In [44]:
```
annual_ytm = semi_annual_ytm*2
annual_ytm
```

Out[44]: 0.03699968484084463

3

In [45]:
```
irs = pd.read_csv('Hw3_Interest_Rates.csv', index_col = 0, parse_da
irs = pd.DataFrame(irs, columns = ['Euro Area', 'United States'])
df = pd.read_excel('ex_rates.xlsx', index_col = 0, parse_dates = Tr
df.head()
```

Out[45]:

	Australia	Canada	Euro Area	Japan	New Zealand	Norway	Sweden	Switzerland	United Kingdom
2000-02-01	1.627869	1.4496	1.029442	110.18	2.056978	8.3184	8.7150	1.6488	0.627865
2000-03-01	1.651528	1.4494	1.046792	105.85	1.995211	8.4670	8.6700	1.6688	0.626920
2000-04-01	1.692334	1.4801	1.100715	106.55	2.052967	8.9681	8.9275	1.7327	0.637714
2000-05-01	1.743679	1.4965	1.074922	106.65	2.179124	8.9272	8.9950	1.6880	0.668986
2000-06-01	1.670565	1.4806	1.046463	105.40	2.137666	8.5653	8.8125	1.6263	0.661288

In [46]:
```
df['United States'] = 1
```

In [48]:
```
df = pd.DataFrame(df, columns = ['Euro Area', 'United States'])
```

In [49]:
```
# calculate the currencies with the highest and lowest yields each 
maxI = irs.idxmax(axis = 1)
minI = irs.idxmin(axis = 1)
```

In [50]:
```
ex = pd.DataFrame([maxI, minI]).T
ex.columns = ['High Yield', 'Low Yield']
```

In [51]:

```
profits = np.array([])
for j in range(len(df)-1):

    # identify the country with the highest (long)
    # and lowest (short) yield
    long = maxI[j]
    short = minI[j]

    # get the exchange rate at t0 and t+1
    # for the short
    sts0 = df[short][j]
    sts1 = df[short][j+1]

    # get the monthly interest rate
    # for the short
    si = irs[short][j]

    # calculate the amount owed
    owed = 10000*sts0*si/sts1

    # get the exchange rate at t0 and t+1
    # for the long
    stl0 = df[long][j]
    stl1 = df[long][j+1]

    # get the monthly interest rate for the long
    li = irs[long][j]

    # calculate the ending balance
    balance = 10000*stl0*li/stl1

    # calculate the profit
    profit = balance - owed

    # store the profits
    profits = np.append(profits, profit)

    print(profit)


profits = pd.DataFrame(profits, index = irs.index[:-1], columns = [
```

```
20.22916290257288
20.583626053273154
17.976983025857884
17.655060295389333
18.86671396819589
17.905820576733696
15.135396976264737
15.406686050990835
11.580064399237514
9.356143889999515
12.018875740397284
6.9448746462251805
5.762641406511825
1.2567009105616833
-0.4889971739729333
4.943391456299999
6.911464389457869
8.164138351445807
6.92242214674814
```

In [52]:

```
maxI = irs.idxmax(axis = 1)
minI = irs.idxmin(axis = 1)
```

In [53]:

```
profits.idxmax()
```

Out[53]:

```
Profit   2018-12-01
dtype: datetime64[ns]
```

In [54]:

```
Cumulative_Yearly = profits.Profit.resample('Y').sum().cumsum()
rorY = profits.Profit.resample('Y').sum()/10000*100
```

In [55]:

```
# long
nz_0 = 1.616815
nz_1 = 1.538225
nz_I = 1.006029
balance = 10000*nz_0*nz_I/nz_1

# short
j_0 = 114.80
j_1 = 117.32
j_I = 1.000366
owed = 10000*j_0*j_I/j_1

balance - owed
```

Out[55]:

```
785.4996025595319
```

In [56]:
```
cp = profits.resample('Y').sum().cumsum().append(pd.DataFrame(0, in
```

In [58]:
```
fig, ax = plt.subplots(figsize = (15, 5))
plt.title('Cumulative Profit vs. Date')
plt.plot(cp)
plt.ylabel('Cumulative Profit')
plt.xlabel('Date')

plt.grid()
```

Total rate of return:

In [59]:
```
P = 1000
A = 1000 *((cp['Profit'][-1]/100)+1)
T = len(data)/12
ccror = np.log(A/P)/T
ccror
```

Out[59]: 0.032091872198610365

Annualized Return

In [60]:
```
annual = (A/P)**(1/T)-1
annual
```

Out[60]: 0.032612369316050493

Alpha and Beta

In [61]:
```
sp500 = yf.download("^GSPC", start = "2000-02-01" , end = "2023-01-
```

```
[*********************100%***********************]  1 of 1 completed
```

In [62]:
```
rf_rate = pd.read_csv('1-year-treasury-rate-yield-chart.csv', index
#subtest
```

In [63]:
```
# match index and dates with our data
drange = pd.date_range(start = '02/2000', end = '01/2023', freq = '
profits.index = drange
profits['rf'] = rf_rate
```

In [64]:
```
# add the monthly risk free rate to our data
profits['rf'] = rf_rate
profits['rf'] = profits['rf'].ffill()

sp500.reset_index(drop=True, inplace=True)
sp500.index = drange
```

In [65]:
```
profits['SP500'] = sp500['Adj Close']
```

In [66]:
```
profits['Profit_excess'] = profits['Profit']-profits['rf']

profits['market_excess'] = profits['SP500']-profits['rf']
```

In [67]:
```
smf.ols('Profit_excess~market_excess', data = profits).fit().summar
```

Out[67]:

OLS Regression Results

Dep. Variable:	Profit_excess	**R-squared:**	0.064
Model:	OLS	**Adj. R-squared:**	0.060
Method:	Least Squares	**F-statistic:**	18.52
Date:	Wed, 08 Mar 2023	**Prob (F-statistic):**	2.34e-05
Time:	23:12:17	**Log-Likelihood:**	-857.56
No. Observations:	275	**AIC:**	1719.
Df Residuals:	273	**BIC:**	1726.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	4.7666	0.723	6.596	0.000	3.344	6.189
market_excess	0.0015	0.000	4.304	0.000	0.001	0.002

Omnibus:	28.810	**Durbin-Watson:**	0.123
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	12.312
Skew:	0.299	**Prob(JB):**	0.00212
Kurtosis:	2.154	**Cond. No.**	4.60e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.6e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

Sharpe Ratio

In [68]:
```
rorY.mean()/rorY.std()
```

Out[68]: 1.5672049992416297

Gini Coefficient

In [69]:
```
def GINI_COEFF(returns):
    # get the number of periods -> will allow us to calculate the a
    periods = len(returns)

    # sort values and sum to calculate the Lorenz curve
    LorenzCurve = np.cumsum(returns.sort_values(by = 'Profit'))

    # start from 0:
    LorenzCurve = pd.DataFrame({'Profit': [0]}).append(LorenzCurve)
    Line = LorenzCurve.copy()
    # form the line that encompasses A and B
    Line['Profit'] = np.arange(0, 1+1/periods, 1/periods) * max(Lor

    # calculate the area of A+B
    UpArea = 0
    for i in range(1, len(returns)):
        UpArea = UpArea + ((Line.iloc[i, :] - LorenzCurve.iloc[i, :
                            + Line.iloc[i-1, :] - LorenzCurve.iloc[

    #calculate the area of A+B+C
    if min(LorenzCurve['Profit']) < 0:
        AllArea = ((np.abs(min(LorenzCurve['Profit'])) * periods) +
        ((max(LorenzCurve['Profit']) * periods)/2))
    else:
        AllArea = ((max(LorenzCurve['Profit']) * periods)/2)

    gini = UpArea/AllArea
    return  gini[0]
```

In [70]:
```
returns = profits[['Profit']]
returns.columns = ['Profit']
GINI_COEFF(returns)
```

Out[70]: 0.3942283532171351

4

a

It means that there is a risk of investors not believing the Fed in the upcoming months, and that could make monetary policy less effective. This comes from the Fed's response to investors, where they explained that they are paying attention to the market and the incoming data to implement their policies.

b

Stock prices will increase.

5

The alternative explanation is that the recession threats have receeded due to the economy's capacity to recover from external shocks.