# Youssef Mahmoud 905854027

In [1]:
```
import warnings
warnings.filterwarnings('ignore')

import pmdarima as pm
import scipy.stats as st
import yfinance as yf
import matplotlib.pyplot as plt
import pandas as pd
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
from fredapi import Fred
```

## 1 - Trading Strategy

In [2]:
```
data = pd.read_excel('Bull_Bear_Spread.xlsx', parse_dates = True, i
```

In [3]:
```
data.head()
```

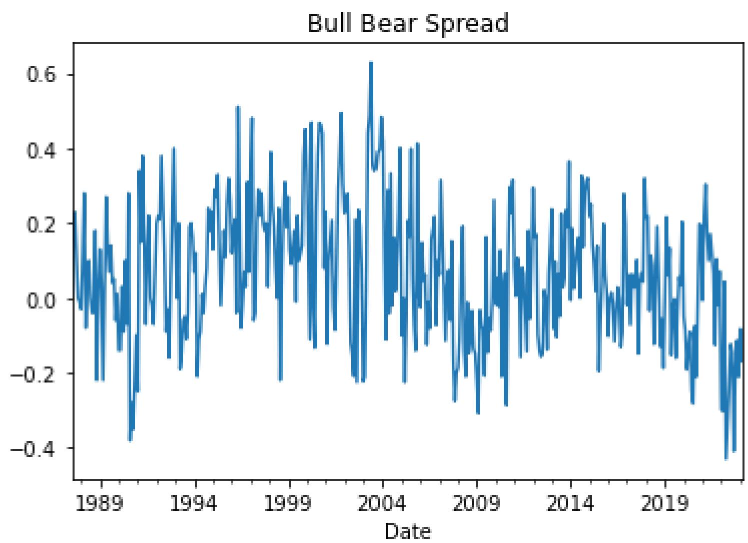
Out[3]:

| | Close |
|---|---|
| **Date** | |
| **1987-07-24** | 0.22 |
| **1987-07-31** | 0.00 |
| **1987-08-07** | 0.27 |
| **1987-08-14** | 0.25 |
| **1987-08-21** | 0.60 |

In [4]:
```
# pull in prices
sector = '^GSPC'
sector = yf.download(sector)[['Adj Close']].copy()
data2 = pd.merge_asof(sector, data, left_index = True, right_index =
data2 = data2.resample('M').last()
data2['returns'] = np.log(data2['Adj Close']).diff()
data2.dropna(inplace = True)
data2.columns = ['sp500', 'bb_index', 'sp500_returns']
```

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

In [5]:
```
data2['bb_index'].dropna().plot()
plt.title('Bull Bear Spread')
plt.show()
```



In [6]:

```
k = 0.07
w = 40
z = 2

# This implements the kalman filter in python
# It is simple otherwise to create using a for loop
data2['Filter'] = data2.bb_index.ewm(alpha = k, adjust = False).mea

# Compute the filter error
data2['Filter Error'] = data2.bb_index - data2['Filter']

# compute the rolling standard deviation
data2['std'] = data2['Filter Error'].rolling(w).std()

# create our confidence intervals or "boundaries of inaction"
# these are scaled by teh number of standard deviations "z"
data2['Upper'] = data2['Filter'] + z*data2['std']
data2['Lower'] = data2['Filter'] - z*data2['std']

# Create signal that evaluates whether we are outside the threshold
# then multiply by the direction of the mistake
# (we use economic theory to decide which direction is long or shor
data2['test'] = np.where(data2['Filter Error'].abs()>z*data2['std']
```

In [7]:

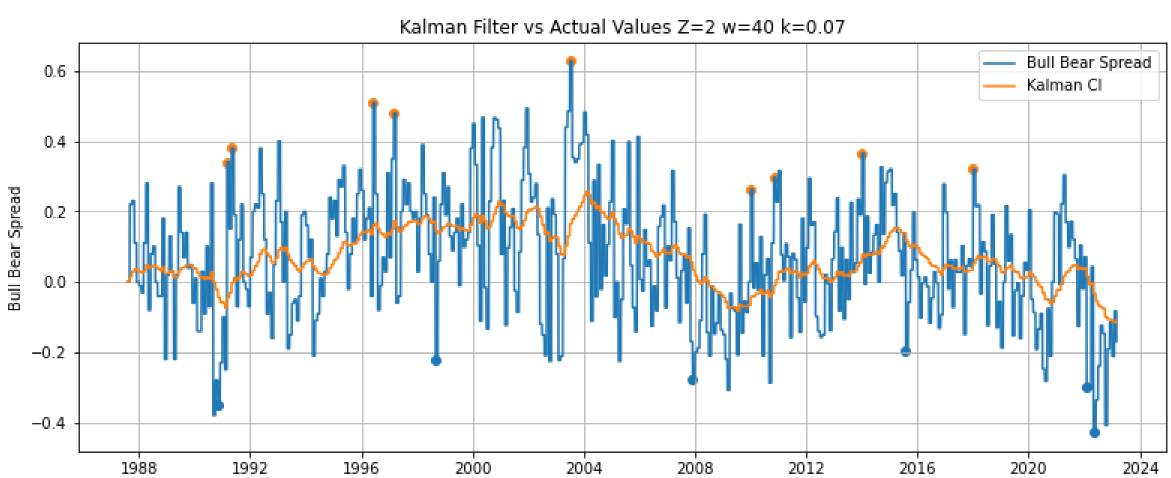
```
# Create a dataframe at a daily frequency wiyh start and end
# dates that cover the observation period
drange = pd.date_range(start = data2.index[0], end = data2.index[-2
daily  = pd.DataFrame(index = drange)

# Integrate the monthly dta into the daily data
daily['test'] = data2['test']

daily['Upper'] = data2['Upper']
daily['Lower'] = data2['Lower']
daily['Filter'] = data2['Filter']
daily['bb_index'] = data2['bb_index']

# Fill NA values with the last available value
daily['Upper'] = daily['Upper'].ffill()
daily['Lower'] = daily['Lower'].ffill()
daily['Filter'] = daily['Filter'].ffill()
daily['bb_index'] = daily['bb_index'].ffill()

# fill the remaining NA values with 0's
# also populates the test column
daily = daily.fillna(0)
```

In [8]:

```
# We let the holding period (i) be 15 days
i = 15

# create a new column that we will populate with our daily position
daily.loc[:, str(i)+'_signal'] = 0

# loop through each day in the dataset
for j in daily.index:
    # If our monthly signal is not 0
    if daily.loc[j, 'test'] != 0:
        # Make the next i days equal to the monthly signal
        daily.loc[j:j+datetime.timedelta(i), str(i)+'_signal'] = da

# Below is the holding period I use for the CI strategy
i = 171
daily.loc[:, str(i)+'_signal'] = 0
for j in daily.index:
    if daily.loc[j, 'test'] != 0:
        daily.loc[j:j+datetime.timedelta(i), str(i)+'_signal'] = da
```

In [9]:

```
data2 = daily.dropna()
fig, ax = plt.subplots(figsize = (13, 5))
ax.set_title('Kalman Filter vs Actual Values ' + 'Z='+str(z) + ' w=

ax.set_ylabel('Bull Bear Spread')

# Plot the actual series and the filter
ax.plot(data2['bb_index'])
ax.plot(data2['Filter'])

# add scatterplots using boolean indexing
# We change the colors and shapes based on the conditions
ax.scatter(data2[data2.test == 1].index, data2[data2.test == 1]['bb_
ax.scatter(data2[data2.test == -1].index, data2[data2.test == -1]['
ax.legend(['Bull Bear Spread', 'Kalman CI'])

# this code can let us zoom in on certain time periods
#plt.xlim([datetime.date(2022, 1, 1), datetime.date(2023, 1, 1)])
ax.grid()
```

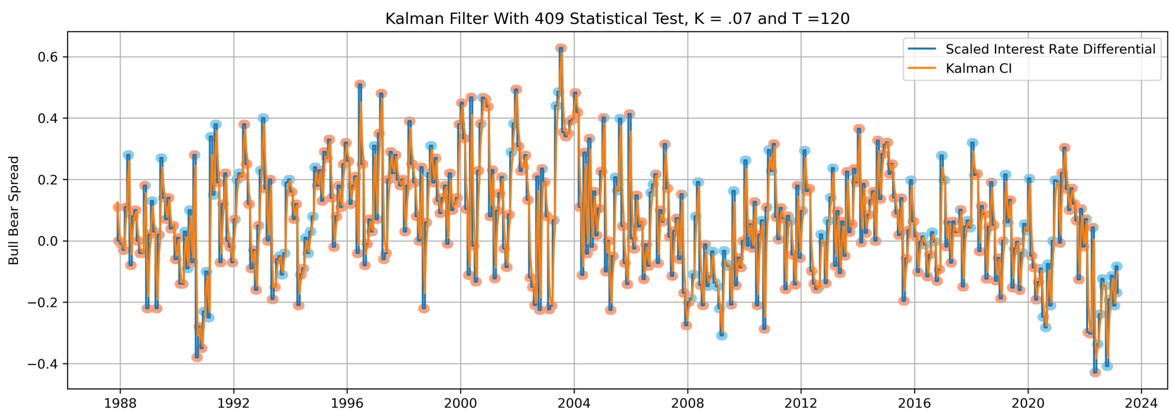


In [10]:
```python
# add the filter gain, critical value, and period for calculating t
# current mean
k = 0.07
z = 2
T = 120
t2 = 100

# This implements the kalman filter in python
# It is simple otherwise to create using a for loop
data2['Filter'] = data2.bb_index.ewm(alpha = k, adjust = False).mean

# The filter error is the difference between the observed value and
data2['Filter Error'] = data2.bb_index - data2['Filter']
```

In [11]:
```python
data22 = data2[['bb_index', 'Filter', 'Filter Error']].copy()

# get the sample mean at each point in time using an expanding wind
data22['E_bar'] = data22['Filter Error'].expanding(T).mean()

# get the sample mean at each point in time using a rolling window
data22['mu_t'] = data22['bb_index'].rolling(T).mean()
data22['mu_t2'] = data22['bb_index'].rolling(T-t2).mean()

# get the variance at each point in time using an expanding window
data22['var_t'] = data22['Filter Error'].expanding(T).std()

# calculate the test statistic
data22['Test Statistic'] = data22['mu_t']/(data22['var_t']/np.sqrt(

# create a new coluimn with a 0 default value
data22['Signal'] = 0
for i in data22.index:
    # create signals based on the sign of the current mean and if w
    if (data22.loc[i, 'mu_t2']/z <= data22.loc[i, 'mu_t']):
        data22.loc[i, 'Signal'] = -1
    elif (data22.loc[i, 'mu_t2']/z > data22.loc[i, 'mu_t']):
        data22.loc[i, 'Signal'] = 1

data22.dropna(inplace = True)
```

In [12]:

```
#interest2 = interest2[interest2.index.year >= 2017]
fig, ax = plt.subplots(figsize = (15, 5), dpi = 300)
ax.set_title('Kalman Filter With 409 Statistical Test, K = .07 and 

ax.set_ylabel('Bull Bear Spread')
ax.plot(data22['bb_index'])
ax.plot(data22['Filter'])


ax.scatter(data22[data22.Signal == 1].index, data22[data22.Signal ==
ax.scatter(data22[data22.Signal == -1].index, data22[data22.Signal =

ax.legend(['Scaled Interest Rate Differential', 'Kalman CI'])


#plt.xlim([datetime.date(2022, 1, 1), datetime.date(2023, 1, 1)])
ax.grid()
```




This contrarian strategy calculates the average sentiment of investors in the last 120 days, and compares the sentiment in the last 20 days. If the sentiment in the last 120 days is higher than in the last 20 days, we enter a short position, since we predict that the market has reached a local maximum. If the sentiment in the last 120 days is lower than in the last 20 days, we enter a long position, for we predict the market has reached a local minimum.

## 2. Signals

In [13]:

```
drange = pd.date_range(start = data22.index[0], end = data22.index[
daily  = pd.DataFrame(index = drange)

daily['test'] = data2['test']

daily['Signal'] = data22['Signal']

daily['Filter'] = data22['Filter']

daily['Filter'] = daily['Filter'].ffill()

daily['bb_index'] = data22['bb_index']
daily['bb_index'] = daily['bb_index'].ffill()
daily = daily.fillna(0)

i = 30
daily[str(i)+'_signal'] = 0
for j in daily.index:
    if daily.loc[j,'Signal'] != 0:
        daily.loc[j:j+datetime.timedelta(i), str(i)+'_signal'] = da
```

In [14]:

```
data22 = pd.merge_asof(sector, data22, left_index = True, right_ind
```

In [15]:

```
drange = pd.date_range(start = data22.index[0], end = data22.index[
exdf = pd.DataFrame(index = drange)

exdf['sp500'] = data22['Adj Close']
#exdf['sp500'] = exdf['sp500'].ffill()

daily['sp500'] = exdf['sp500']
daily['Returns'] = np.log(daily['sp500']).diff()
```

In [16]:

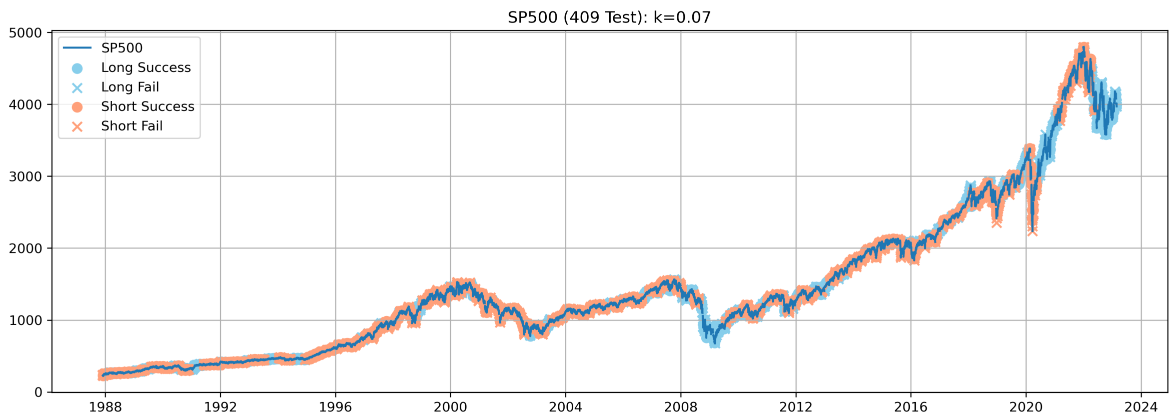
```
s = i
daily[str(s)+'_returns'] = (np.exp((daily[str(i)+'_signal'].shift()
```

In [17]:

```
daily[str(s)+'_success'] = ((daily[daily.Signal!= 0][str(s)+'_retur
```

In [18]:

```
fig, ax = plt.subplots(figsize = (15, 5), dpi = 300)

plt.title('SP500 (409 Test):'+ ' k=' + str(k))

plt.plot(daily['sp500'])

longsuccess = daily[(daily[str(s)+'_success'] == 1) & (daily['Signal
longfail = daily[(daily[str(s)+'_success'] == 0) & (daily['Signal']
shortsuccess = daily[(daily[str(s)+'_success'] == 1) & (daily['Signa
shortfail = daily[(daily[str(s)+'_success'] == 0) & (daily['Signal'

plt.scatter(longsuccess.index, longsuccess['sp500'], color = 'skyblu
plt.scatter(longfail.index, longfail['sp500'], color = 'skyblue', s

plt.scatter(shortsuccess.index, shortsuccess['sp500'], color = 'ligh
plt.scatter(shortfail.index, shortfail['sp500'], color = 'lightsalmo

plt.legend(['SP500', 'Long Success', 'Long Fail', 'Short Success',
#plt.xlim([datetime.date(2022, 1, 1), datetime.date(2023, 1, 1)])
plt.grid()
```




### Equity Curve

In [19]:

```
mret = str(i)+'_returns'
```

In [20]:

```
fig, ax = plt.subplots(figsize = (15, 5))
d2 = daily[daily.index.year >= 2020]
(daily[mret].dropna()).plot()

plt.title('SP500 Contrarian Strategy:'+ ' k=' + str(k))

plt.legend(['Equity Curve', 'Long Success', 'Long Fail', 'Short Succ
plt.ylabel('%')

plt.grid()
```

## Part B

In [21]:
```
#daily = daily.dropna()
daily['cumulative_returns'] = daily['Returns'].cumsum()
daily
```

Out[21]:

| | test | Signal | Filter | bb_index | 30_signal | sp500 | Returns | 30_returns | 30_s |
|---|---|---|---|---|---|---|---|---|---|
| **1987-11-27 00:00:00-05:00** | 0.0 | -1 | 0.125246 | 0.110000 | -1 | 240.339996 | NaN | NaN | |
| **1987-11-28 00:00:00-05:00** | 0.0 | -1 | 0.124179 | 0.110000 | -1 | NaN | NaN | NaN | |
| **1987-11-29 00:00:00-05:00** | 0.0 | -1 | 0.123186 | 0.110000 | -1 | NaN | NaN | NaN | |
| **1987-11-30 00:00:00-05:00** | -0.0 | -1 | 0.114563 | 0.000000 | -1 | 230.300003 | NaN | NaN | |
| **1987-12-01 00:00:00-05:00** | 0.0 | -1 | 0.106544 | 0.000000 | -1 | 232.000000 | 0.007355 | -0.007328 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **2023-02-23 00:00:00-05:00** | 0.0 | 1 | -0.103949 | -0.083094 | 1 | 4012.320068 | 0.005315 | -0.303778 | |
| **2023-02-24 00:00:00-05:00** | 0.0 | 1 | -0.102489 | -0.083094 | 1 | 3970.040039 | -0.010593 | -0.311115 | |
| **2023-02-25 00:00:00-05:00** | 0.0 | 1 | -0.101132 | -0.083094 | 1 | NaN | NaN | NaN | |
| **2023-02-26 00:00:00-05:00** | 0.0 | 1 | -0.099869 | -0.083094 | 1 | NaN | NaN | NaN | |
| **2023-02-27 00:00:00-05:00** | 0.0 | 1 | -0.098695 | -0.083094 | 1 | 3982.239990 | NaN | NaN | |

12877 rows × 10 columns

### Total rate of return

In [22]:
```
P = 1000
A = 1000 *((daily['cumulative_returns'].dropna()[-1]/100)+1)
T = len(data)/12
ccror = np.log(A/P)/T
ccror
```

Out[22]: 0.00014264323878713112

### Annualized Return

In [23]:
```
annualized = (A/P)**(1/T)-1
annualized
```

Out[23]: 0.00014265341281771704

### Rate of return only over the days we hold a position

In [24]:
```
np.log(A/P)/(len(daily[daily['Signal'] != 0])/12)
```

Out[24]: 2.0559590835514124e-05

### Sharpe Ratio

In [25]:
```
rf_rate = pd.read_csv('1-year-treasury-rate-yield-chart.csv', index_
```

In [26]:
```
rf_rate = rf_rate.loc['1987-11-27':'2022-12-22',:]
daily = daily.loc['1987-11-27':'2022-12-22',:]
```

In [27]:
```
# match index and dates with our data
drange = pd.date_range(start = '11/27/1987', end = '12/22/2022', fre
daily.index = drange
daily['rf'] = rf_rate
```

In [28]:
```
# add the monthly risk free rate to our data
daily['rf'] = rf_rate
daily['rf'] = daily['rf'].ffill()
```

In [29]:
```
# Subset strategy returns
return_frame = daily[['Returns']].copy().dropna()

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

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

In [30]:
```
# annualized return method (some use the arithmetic return, which i
anualized_excess = ((excess_return+1).prod()**(12/len(daily))-1)*10
```

In [31]:
```
# calculate the anualized standard deviation
excess_ann_std = excess_return.std()*np.sqrt(12)*100
```

In [32]:
```
print('The Sharpe Ratio of our strategy is:', round((anualized_exce
```

```
The Sharpe Ratio of our strategy is: -0.384
```

### Gini Coefficient

In [33]:
```
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 = 'Returns'))

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

    # 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['Returns']) < 0:
        AllArea = ((np.abs(min(LorenzCurve['Returns'])) * periods) +
        ((max(LorenzCurve['Returns']) * periods)/2))
    else:
        AllArea = ((max(LorenzCurve['Returns']) * periods)/2)

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

In [34]:
```
returns = daily[['Returns']].dropna()[:-1]
returns.columns = ['Returns']
```

In [35]:
```
Gini_Coeff(returns)
```

```
Gini Coefficient: 0.7517069574259009
```

## Articles

### a

The article mentions how the increase in interest rates by the Fed caused the Silicon Valley Bank to collapse. Their investment in illiquid assets as 10-year government bonds led them to lose billions of dollars, and their lack of response to investors caused a bank-run.

## b

EURUSD: Increase, after the fall in the financial sector index.

Ten Year US interest rates: Decrease.

SP500: Decrease, after the bank run.

## c

EURUSD: Increased.

Ten Year US interest rates: Decreased.

SP500: Decreased.

## d

The raise in interest rates has affected two markets: the labor market and the financial sector. On the one hand, startups experienced an increase in borrowing costs due to the rising interest rates, which also increased their labor costs. On the other hand, the illiquid investments by SVB prevented them form addressing the startups' liquidity needs by providing them their deposits.