409 & 442B HW 3 Oishika Kar - Copy

March 3, 2024

[ ]:
```
# Import Libraries
import numpy as np # package for scientific computing
import pandas as pd # package for data manipulation
import os # package for communicating with operating system
import statsmodels.formula.api as smf # package for statistical models i.e. OLS
import scipy.stats as st #open-source library used for scientific and technical␣
↪computing
import matplotlib.pyplot as plt #plotting library for creating visualizations␣
↪in Python
import seaborn as sns #import seaborn as sns
```

0.0.1 1) Use the hw3 data.csv file posted on the website for the following questions. The file contains the PE ratios and Returns for the Industrial and Financial Sectors.

[ ]:
```
# Load the data
data = pd.read_csv('hw3_data.csv', parse_dates = True)
data.head()
```

[ ]:
```
         Date  Health_PE  HealthSector_Returns  Industrial_PE  \
0  1991-04-08    22.0549             -0.014480        15.4770
1  1991-04-09    21.7379             -0.003043        15.2208
2  1991-04-10    21.6718              0.014624        15.2860
3  1991-04-11    21.9911              0.005691        15.4244
4  1991-04-12    22.1166              0.003776        15.4863

   IndustrialSector_Returns
0                 -0.016236
1                 -0.008287
2                  0.019911
3                 -0.001905
4                  0.002857
```

0.0.2 a) Perform a fisher transform on each of the P/E rartios where the transformation window size for Health is 756 observations and 1764 observations for the Industrial Sector

[ ]:
```
#Rolling window size
s_h = 756
s_i= 1764
```

[ ]:
```
# Establish rolling min and max for health:
data['roll_min_h'] = data[['Health_PE']].rolling(int(s_h)).min()
data['roll_max_h'] = data[['Health_PE']].rolling(int(s_h)).max()

# Establish rolling min and max for industrial:
data['roll_min_i'] = data[['Industrial_PE']].rolling(int(s_i)).min()
data['roll_max_i'] = data[['Industrial_PE']].rolling(int(s_i)).max()

#Fisher Transfomation:
data['fisher_health'] = (data['Health_PE'] - data['roll_min_h']) /␣
 ↪(data['roll_max_h'] - data['roll_min_h'])
data['fisher_industrial'] = (data['Industrial_PE'] - data['roll_min_i']) /␣
 ↪(data['roll_max_i'] - data['roll_min_i'])
```

[ ]:
```
data.tail()
```

[ ]:

	Date	Health_PE	HealthSector_Returns	Industrial_PE
8263	2024-01-26	19.8396	0.006819	21.2431
8264	2024-01-29	19.9753	0.002470	21.3780
8265	2024-01-30	20.0247	-0.001112	21.3879
8266	2024-01-31	19.9998	0.012589	21.1433
8267	2024-02-01	20.2532	0.001508	21.5030

	IndustrialSector_Returns	roll_min_h	roll_max_h	roll_min_i
8263	0.003013	14.4757	20.1108	11.5304
8264	0.011887	14.4757	20.1108	11.5304
8265	-0.012228	14.4757	20.1108	11.5304
8266	0.000915	14.4757	20.1108	11.5304
8267	0.007644	14.4757	20.2532	11.5304

	roll_max_i	fisher_health	fisher_industrial
8263	53.7319	0.951873	0.230151
8264	53.7319	0.975954	0.233347
8265	53.7319	0.984721	0.233582
8266	53.7319	0.980302	0.227786
8267	53.7319	1.000000	0.236309

0.0.3 b)Write python code that will implement a trading strategy that will enter a short position when the fisher transformed PE of a sector crosses below an upper threshold and go long when we cross above a lower threshold. Once you enter a position, you will only exit when you receive a signal in the opposite direction. (Note: Neither the lecture or the exercise did exactly this, since we entered positions when we crossed the thresholds the first time. You must make changes).

[ ]:
```python
def strategy(u, l, data):
    # Make a copy of the original data set
    data_copy = data.copy()

    # Health sector signals
    data_copy['signal_h'] = np.NaN
    # Enter short position for Health when crossing below upper threshold
    data_copy.loc[(data_copy['fisher_health'].shift() > u) & (data_copy['fisher_health'] < u), 'signal_h'] = -1
    # Enter long position for Health when crossing above lower threshold
    data_copy.loc[(data_copy['fisher_health'].shift() < l) & (data_copy['fisher_health'] > l), 'signal_h'] = 1
    # Forward fill the 'signal_h' column to carry the position forward
    data_copy['signal_h'] = data_copy['signal_h'].ffill()

    # Industrial sector signals
    data_copy['signal_i'] = np.NaN
    # Enter short position for Industrial when crossing below upper threshold
    data_copy.loc[(data_copy['fisher_industrial'].shift() > u) & (data_copy['fisher_industrial']< u), 'signal_i'] = -1
    # Enter long position for Industrial when crossing above lower threshold
    data_copy.loc[(data_copy['fisher_industrial'].shift() < l) & (data_copy['fisher_industrial']> l), 'signal_i'] = 1
    # Forward fill the 'signal_i' column to carry the position forward
    data_copy['signal_i'] = data_copy['signal_i'].ffill()

    return data_copy
```

[ ]:
```python
# Define upper and lower thresholds
upper_threshold = 0.8
lower_threshold = 0.2

# Testing the trading strategy
result_data = strategy(upper_threshold, lower_threshold, data)

# Display the resulting DataFrame
result_data
```

[ ]:

```
            Date  Health_PE  HealthSector_Returns  Industrial_PE  \
0     1991-04-08    22.0549             -0.014480        15.4770
1     1991-04-09    21.7379             -0.003043        15.2208
2     1991-04-10    21.6718              0.014624        15.2860
3     1991-04-11    21.9911              0.005691        15.4244
4     1991-04-12    22.1166              0.003776        15.4863
…            …        …                    …             …
8263  2024-01-26    19.8396              0.006819        21.2431
8264  2024-01-29    19.9753              0.002470        21.3780
8265  2024-01-30    20.0247             -0.001112        21.3879
8266  2024-01-31    19.9998              0.012589        21.1433
8267  2024-02-01    20.2532              0.001508        21.5030

      IndustrialSector_Returns  roll_min_h  roll_max_h  roll_min_i  \
0                    -0.016236         NaN         NaN         NaN
1                    -0.008287         NaN         NaN         NaN
2                     0.019911         NaN         NaN         NaN
3                    -0.001905         NaN         NaN         NaN
4                     0.002857         NaN         NaN         NaN
…                          …         …         …         …
8263                  0.003013     14.4757     20.1108     11.5304
8264                  0.011887     14.4757     20.1108     11.5304
8265                 -0.012228     14.4757     20.1108     11.5304
8266                  0.000915     14.4757     20.1108     11.5304
8267                  0.007644     14.4757     20.2532     11.5304

      roll_max_i  fisher_health  fisher_industrial  signal_h  signal_i
0            NaN            NaN                NaN       NaN       NaN
1            NaN            NaN                NaN       NaN       NaN
2            NaN            NaN                NaN       NaN       NaN
3            NaN            NaN                NaN       NaN       NaN
4            NaN            NaN                NaN       NaN       NaN
…            …            …                …       …       …
8263     53.7319       0.951873           0.230151       1.0       1.0
8264     53.7319       0.975954           0.233347       1.0       1.0
8265     53.7319       0.984721           0.233582       1.0       1.0
8266     53.7319       0.980302           0.227786       1.0       1.0
8267     53.7319       1.000000           0.236309       1.0       1.0

[8268 rows x 13 columns]
```

0.0.4 c) Test at least 400 different combinations of valid (Upper threshold greater than or equal to lower threshold) hyperparameters for your boundaries for each fisher transformed series of PEs. Use the geometric mean of returns as your metric for the success of each outcome.

[ ]:
```python
def geometric_mean(data_copy): #simple returns has been assumed
    data_copy["return_health"] = (data_copy["signal_h"].shift() *␣
↪data_copy['HealthSector_Returns'])
    data_copy["return_health"] = data_copy["return_health"].fillna(0)
    data_copy["return_industrial"] = (data_copy["signal_i"].shift() *␣
↪data_copy['IndustrialSector_Returns'])
    data_copy["return_industrial"] = data_copy["return_industrial"].fillna(0)

    GM_h = ((data_copy['return_health']+1).cumprod().iloc[-1])**(1/
↪len(data_copy['return_health'])) - 1
    GM_i = ((data_copy['return_industrial']+1).cumprod().iloc[-1])**(1/
↪len(data_copy['return_industrial'])) - 1
    return GM_h, GM_i
```

[ ]:
```python
u1_values = np.arange(0, 1, 0.05)  #number of points to ensure at least 400␣
↪combinations
l1_values = np.arange(0, 1, 0.05)

results = []

result_data_copy = data.copy()

for u1 in u1_values:
    for l1 in filter(lambda x: x < u1, l1_values):
        print(u1, l1)
        data_copy = strategy(u1, l1, result_data_copy.copy())  # Making sure to␣
↪pass a copy of the original data
        GM_h, GM_i = geometric_mean(data_copy)
        results.append({'u1': u1, 'l1': l1, 'GM_h': GM_h, 'GM_i': GM_i})

results_data = pd.DataFrame(results)

sorted_results = results_data.sort_values(by=['GM_h', 'GM_i'], ascending=False)
sorted_results = sorted_results.dropna()

sorted_results["GM_h"] = sorted_results["GM_h"] * 10000
sorted_results["GM_i"] = sorted_results["GM_i"] * 10000
```

```
0.05 0.0
0.1 0.0
0.1 0.05
0.15000000000000002 0.0
0.15000000000000002 0.05
```

```
0.15000000000000002 0.1
0.2 0.0
0.2 0.05
0.2 0.1
0.2 0.15000000000000002
0.25 0.0
0.25 0.05
0.25 0.1
0.25 0.15000000000000002
0.25 0.2
0.30000000000000004 0.0
0.30000000000000004 0.05
0.30000000000000004 0.1
0.30000000000000004 0.15000000000000002
0.30000000000000004 0.2
0.30000000000000004 0.25
0.35000000000000003 0.0
0.35000000000000003 0.05
0.35000000000000003 0.1
0.35000000000000003 0.15000000000000002
0.35000000000000003 0.2
0.35000000000000003 0.25
0.35000000000000003 0.30000000000000004
0.4 0.0
0.4 0.05
0.4 0.1
0.4 0.15000000000000002
0.4 0.2
0.4 0.25
0.4 0.30000000000000004
0.4 0.35000000000000003
0.45 0.0
0.45 0.05
0.45 0.1
0.45 0.15000000000000002
0.45 0.2
0.45 0.25
0.45 0.30000000000000004
0.45 0.35000000000000003
0.45 0.4
0.5 0.0
0.5 0.05
0.5 0.1
0.5 0.15000000000000002
0.5 0.2
0.5 0.25
0.5 0.30000000000000004
0.5 0.35000000000000003
```

0.5 0.4
0.5 0.45
0.55 0.0
0.55 0.05
0.55 0.1
0.55 0.15000000000000002
0.55 0.2
0.55 0.25
0.55 0.30000000000000004
0.55 0.35000000000000003
0.55 0.4
0.55 0.45
0.55 0.5
0.6000000000000001 0.0
0.6000000000000001 0.05
0.6000000000000001 0.1
0.6000000000000001 0.15000000000000002
0.6000000000000001 0.2
0.6000000000000001 0.25
0.6000000000000001 0.30000000000000004
0.6000000000000001 0.35000000000000003
0.6000000000000001 0.4
0.6000000000000001 0.45
0.6000000000000001 0.5
0.6000000000000001 0.55
0.65 0.0
0.65 0.05
0.65 0.1
0.65 0.15000000000000002
0.65 0.2
0.65 0.25
0.65 0.30000000000000004
0.65 0.35000000000000003
0.65 0.4
0.65 0.45
0.65 0.5
0.65 0.55
0.65 0.6000000000000001
0.7000000000000001 0.0
0.7000000000000001 0.05
0.7000000000000001 0.1
0.7000000000000001 0.15000000000000002
0.7000000000000001 0.2
0.7000000000000001 0.25
0.7000000000000001 0.30000000000000004
0.7000000000000001 0.35000000000000003
0.7000000000000001 0.4
0.7000000000000001 0.45

```
0.7000000000000001 0.5
0.7000000000000001 0.55
0.7000000000000001 0.6000000000000001
0.7000000000000001 0.65
0.75 0.0
0.75 0.05
0.75 0.1
0.75 0.15000000000000002
0.75 0.2
0.75 0.25
0.75 0.30000000000000004
0.75 0.35000000000000003
0.75 0.4
0.75 0.45
0.75 0.5
0.75 0.55
0.75 0.6000000000000001
0.75 0.65
0.75 0.7000000000000001
0.8 0.0
0.8 0.05
0.8 0.1
0.8 0.15000000000000002
0.8 0.2
0.8 0.25
0.8 0.30000000000000004
0.8 0.35000000000000003
0.8 0.4
0.8 0.45
0.8 0.5
0.8 0.55
0.8 0.6000000000000001
0.8 0.65
0.8 0.7000000000000001
0.8 0.75
0.8500000000000001 0.0
0.8500000000000001 0.05
0.8500000000000001 0.1
0.8500000000000001 0.15000000000000002
0.8500000000000001 0.2
0.8500000000000001 0.25
0.8500000000000001 0.30000000000000004
0.8500000000000001 0.35000000000000003
0.8500000000000001 0.4
0.8500000000000001 0.45
0.8500000000000001 0.5
0.8500000000000001 0.55
0.8500000000000001 0.6000000000000001
```

```
0.8500000000000001 0.65
0.8500000000000001 0.7000000000000001
0.8500000000000001 0.75
0.8500000000000001 0.8
0.9 0.0
0.9 0.05
0.9 0.1
0.9 0.15000000000000002
0.9 0.2
0.9 0.25
0.9 0.30000000000000004
0.9 0.35000000000000003
0.9 0.4
0.9 0.45
0.9 0.5
0.9 0.55
0.9 0.6000000000000001
0.9 0.65
0.9 0.7000000000000001
0.9 0.75
0.9 0.8
0.9 0.8500000000000001
0.9500000000000001 0.0
0.9500000000000001 0.05
0.9500000000000001 0.1
0.9500000000000001 0.15000000000000002
0.9500000000000001 0.2
0.9500000000000001 0.25
0.9500000000000001 0.30000000000000004
0.9500000000000001 0.35000000000000003
0.9500000000000001 0.4
0.9500000000000001 0.45
0.9500000000000001 0.5
0.9500000000000001 0.55
0.9500000000000001 0.6000000000000001
0.9500000000000001 0.65
0.9500000000000001 0.7000000000000001
0.9500000000000001 0.75
0.9500000000000001 0.8
0.9500000000000001 0.8500000000000001
0.9500000000000001 0.9
```

[ ]:
```
sorted_results.dropna()
```

[ ]:

	u1	l1	GM_h	GM_i
27	0.35	0.30	0.873051	-2.507838
14	0.25	0.20	0.822049	-0.425975

```
77   0.60  0.55  0.769378 -0.727629
33   0.40  0.25  0.698901 -2.155314
34   0.40  0.30  0.609279 -2.016938
..    …     …         …         …
21   0.35  0.00 -4.019089 -2.231039
3    0.15  0.00 -4.207418 -2.454894
15   0.30  0.00 -4.232335 -2.221700
6    0.20  0.00 -4.266324 -2.410998
10   0.25  0.00 -4.284904 -2.239164

[190 rows x 4 columns]
```

[ ]:
```python
from IPython.display import display, HTML
```

0.0.5 Q2) Hyperparameter analysis.

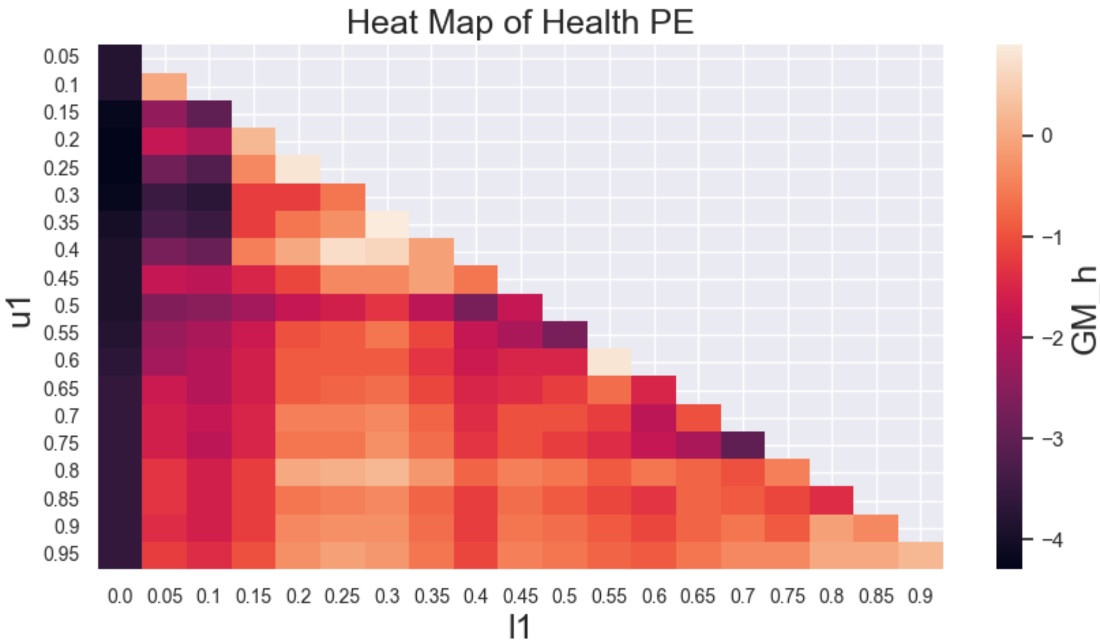
[ ]:
```python
max_GM_h_index = sorted_results['GM_h'].idxmax()
max_GM_i_index = sorted_results['GM_i'].idxmax()

# Extract the corresponding hyperparameters and GM values
best_hyperparameters_h = sorted_results.loc[max_GM_h_index, ['u1', 'l1',␣
↪'GM_h']]
best_hyperparameters_i = sorted_results.loc[max_GM_i_index, ['u1', 'l1',␣
↪'GM_i']]

display(HTML("<b>Best Hyperparameters for Health:</b>"))
display(best_hyperparameters_h)

display(HTML("<b>Best Hyperparameters for Industrial:</b>"))
display(best_hyperparameters_i)
```

```
<IPython.core.display.HTML object>

u1      0.350000
l1      0.300000
GM_h    0.873051
Name: 27, dtype: float64

<IPython.core.display.HTML object>

u1      0.700000
l1      0.550000
GM_i    2.035046
Name: 102, dtype: float64
```

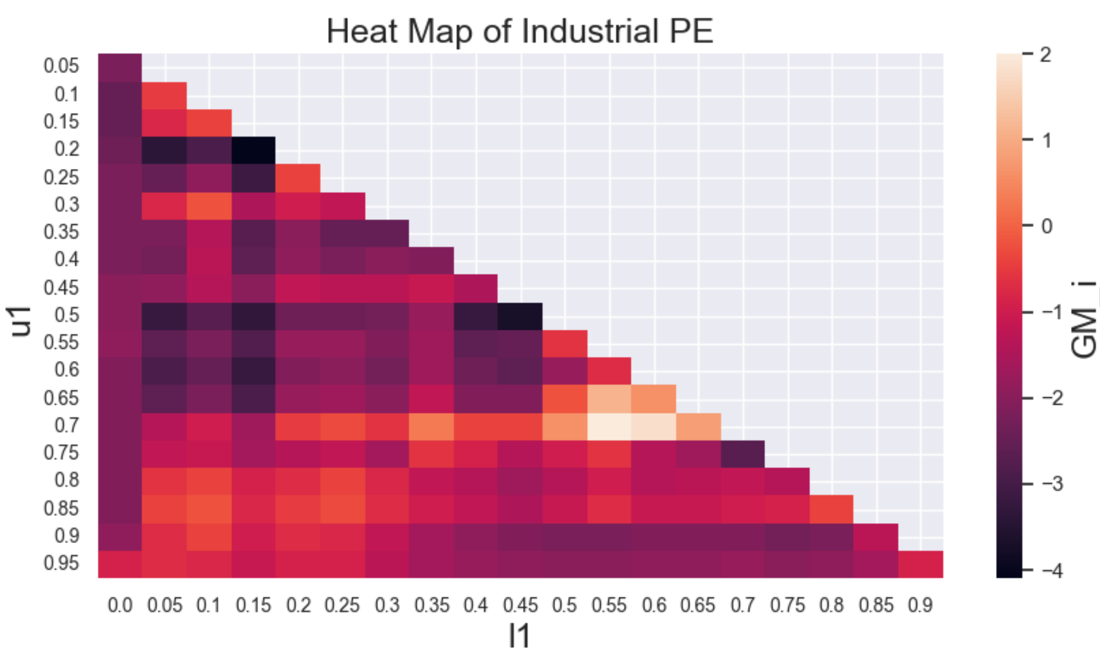
0.0.6 a) Create two properly labelled heat maps (with a clear distinction between the values of the heat map, scaling may be required) for your tested hyper parameter values in the industrial and health sectors.

[ ]:
```python
def heatmap(x, y, metric, values, title):

    # 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]).mean(),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=(10, 5))
    ax = sns.heatmap(heat, fmt='.1g')
    ax.set_title(title,size = 18)
    ax.tick_params(axis='both', which='major', labelsize=10)
    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()
```

[ ]:
```python
tmp = sorted_results
heatmap("u1", "l1","GM_h", tmp,'Heat Map of Health PE')
```




```
[ ]: tmp = sorted_results
     heatmap("u1", "l1","GM_i", tmp,'Heat Map of Industrial PE')
```



0.0.7 b) What do the heatmaps tell you about the hyperparameters that are best for each sector? Is there any similarity between the two?

Best hyprerparameters sector wise - For Health sector, u1 = 0.35 and l1 = 0.30 For Industrial sector u1 = 0.70 and l1 = 0.55

For health sector u1 and l1 are almost same. For industrial sector, there appears to be a significant difference between u1 and l1. In case of health sector when u1 and l1 both are low, GM is highest. In case of industrial sector, when u1 and l1 are increasing slightly in magnitude, we get highest GM.

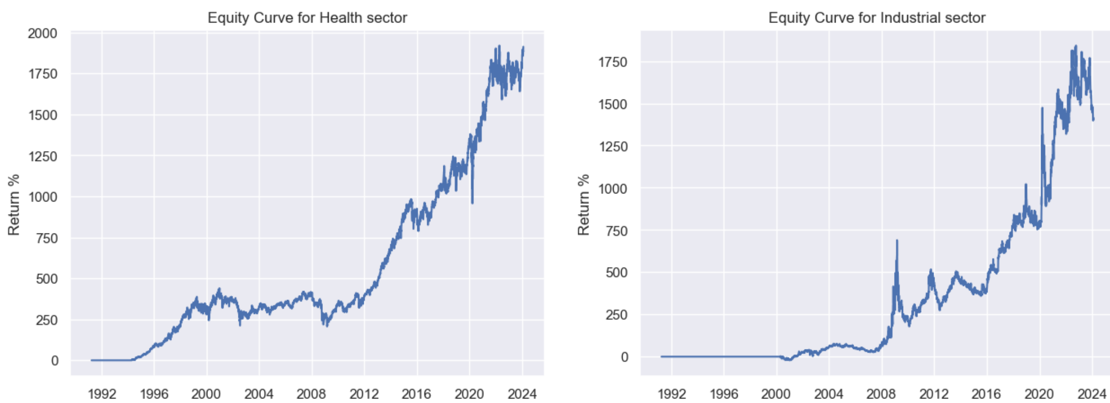
[ ]:
```
data.columns
```

[ ]:
```
Index(['Date', 'Health_PE', 'HealthSector_Returns', 'Industrial_PE',
       'IndustrialSector_Returns', 'roll_min_h', 'roll_max_h', 'roll_min_i',
       'roll_max_i', 'fisher_health', 'fisher_industrial'],
      dtype='object')
```

0.0.8 c) Based on what you learned from the heatmaps, pick a pair of hyperparameters for the health and industrial sector strategies and visualize the equity curve they produce for each.

[ ]:
```
u1, l1 = 1, 0.3
u2, l2 = 0.7, 0.55

# Health sector strategy
data_copy_new_h = strategy(u1, l1, data.copy())
data_copy_new_h['Date'] = pd.to_datetime(data_copy_new_h['Date'])
data_copy_new_h.set_index('Date', inplace=True)
data_copy_new_h["strat_returns_h"] = (data_copy_new_h["signal_h"].shift() *␣
 ↪data_copy_new_h['HealthSector_Returns'])
data_copy_new_h["strat_returns_h"] = data_copy_new_h["strat_returns_h"].
 ↪fillna(0)
data_copy_new_h["cumulative_returns_h"] = (np.
 ↪exp(data_copy_new_h['strat_returns_h'].cumsum()) - 1) * 100

# Industrial sector strategy
data_copy_new_i = strategy(u2, l2, data.copy())
data_copy_new_i['Date'] = pd.to_datetime(data_copy_new_i['Date'])
data_copy_new_i.set_index('Date', inplace=True)
data_copy_new_i["strat_returns_i"] = (data_copy_new_i["signal_i"].shift() *␣
 ↪data_copy_new_i['IndustrialSector_Returns'])
data_copy_new_i["strat_returns_i"] = data_copy_new_i["strat_returns_i"].
 ↪fillna(0)
data_copy_new_i["cumulative_returns_i"] = (np.
 ↪exp(data_copy_new_i['strat_returns_i'].cumsum()) - 1) * 100

# Plotting side by side
```

```
plt.figure(figsize=(15, 5))

plt.subplot(1, 2, 1)
plt.plot(data_copy_new_h.cumulative_returns_h)
plt.ylabel('Return %')
plt.title('Equity Curve for Health sector')

plt.subplot(1, 2, 2)
plt.plot(data_copy_new_i.cumulative_returns_i)
plt.ylabel('Return %')
plt.title('Equity Curve for Industrial sector')

plt.show()
```



0.0.9 Q3) Portfolio creation

0.0.10 a) Use the outcomes you generated in 2c to create an equally weighted portfolio.

[ ]:
```
portfolio = pd.DataFrame({"strat_returns": []})

# Combine the strategy returns and calculate the equally weighted portfolio␣
↪returns
portfolio["strat_returns"] = (data_copy_new_h["strat_returns_h"].fillna(0) +␣
↪data_copy_new_i["strat_returns_i"].fillna(0)) / 2
```

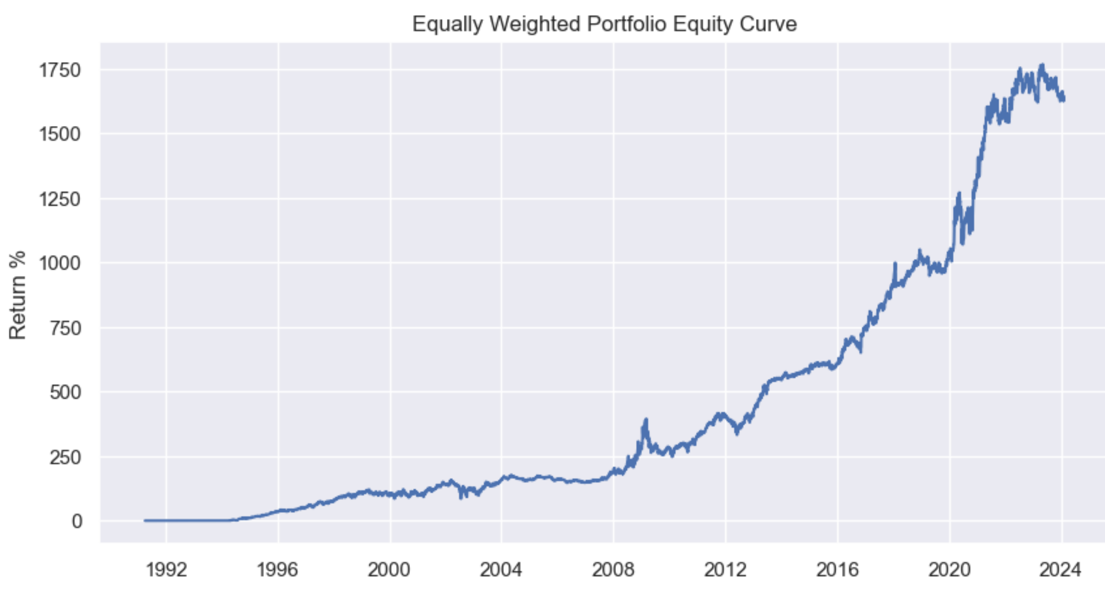
[ ]:
```
portfolio
```

[ ]:
```
            strat_returns
Date
1991-04-08       0.000000
1991-04-09       0.000000
1991-04-10       0.000000
```

```
1991-04-11        0.000000
1991-04-12        0.000000
…                    …
2024-01-26        0.001903
2024-01-29       -0.004708
2024-01-30        0.005558
2024-01-31        0.005837
2024-02-01       -0.003068

[8268 rows x 1 columns]
```

0.0.11 b) Show the equity curve of the portfolio from the previous part.

[ ]:
```
# Calculate the cumulative returns of the equally weighted portfolio
portfolio["cumulative_returns"] = (np.exp(portfolio['strat_returns'].cumsum())␣
↪- 1) * 100

# Plot the equity curve for the equally weighted portfolio
plt.figure(figsize=(10, 5))
plt.plot(portfolio["cumulative_returns"])
plt.ylabel('Return %')
plt.title('Equally Weighted Portfolio Equity Curve')

plt.show()
```



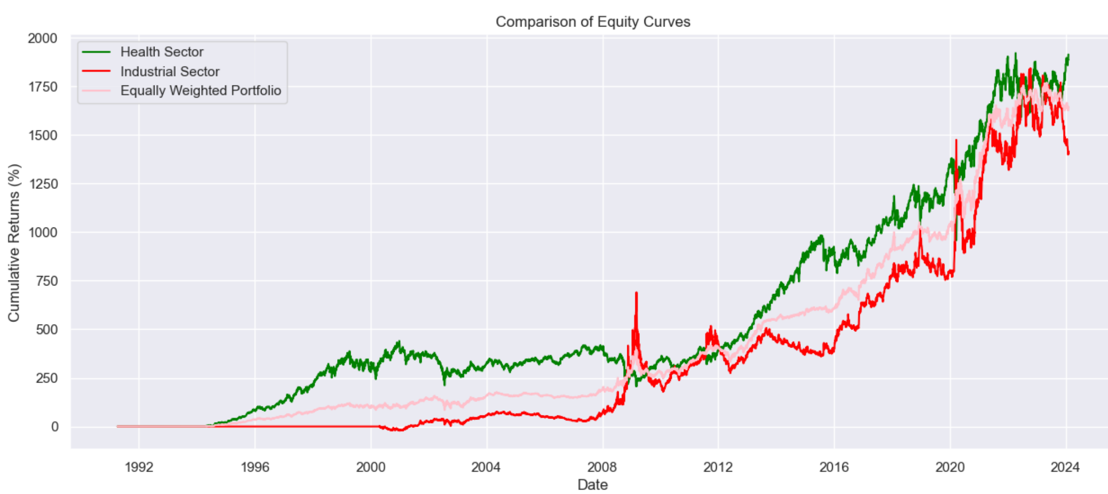
[ ]:
```
# Calculate CCRoR for Health Sector
t_h = len(data_copy_new_h) / 252
```

```python
A_h = data_copy_new_h["cumulative_returns_h"].iloc[-1] + 1
ccror_h = (np.log(A_h) / t_h) * 100

# Calculate CCRoR for Industrial Sector
t_i = len(data_copy_new_i) / 252
A_i = data_copy_new_i["cumulative_returns_i"].iloc[-1] + 1
ccror_i = (np.log(A_i) / t_i) * 100

# Calculate CCRoR for the Equally Weighted Portfolio
t_portfolio = len(portfolio) / 252
A_portfolio = portfolio["cumulative_returns"].iloc[-1] + 1
ccror_portfolio = (np.log(A_portfolio) / t_portfolio) * 100

print("CCRoR:")
print(f"Health Sector: {ccror_h:.2f}%")
print(f"Equally Weighted Portfolio: {ccror_portfolio:.2f}%")
print(f"Industrial Sector: {ccror_i:.2f}%\n")


# Calculate Annualized Returns for Health Sector
AR_h = ((A_h ** (1 / t_h)) - 1) * 100

# Calculate Annualized Returns for Industrial Sector
AR_i = ((A_i ** (1 / t_i)) - 1) * 100

# Calculate Annualized Returns for the Equally Weighted Portfolio
AR_portfolio = ((A_portfolio ** (1 / t_portfolio)) - 1) * 100

print("Annualized Returns:")
print(f"Health Sector: {AR_h:.2f}%")
print(f"Equally Weighted Portfolio:{AR_portfolio:.2f}%")
print(f"Industrial Sector: {AR_i:.2f}%")
```

```
CCRoR:
Health Sector: 23.03%
Equally Weighted Portfolio: 22.56%
Industrial Sector: 22.09%

Annualized Returns:
Health Sector: 25.90%
Equally Weighted Portfolio:25.31%
Industrial Sector: 24.72%
```

The portfolio, though not achieving the same level of risk-adjusted returns as the Health Sector, is able to outperform the Industrial Sector in terms of CCRoR. The balanced nature of the portfolio helps in achieving a risk-return profile between the two sectors. The portfolio, while not reaching the same level of annualized returns as the Health Sector, still provides a higher return compared to the Industrial Sector. The diversification benefits of the portfolio contribute to achieving a balance in overall annualized returns

[ ]:
```
# Plotting the Cumulative Returns for each sector and the Equally Weighted␣
↪Portfolio
plt.figure(figsize=(15, 6))

# Health Sector
plt.plot(data_copy_new_h.index, data_copy_new_h['cumulative_returns_h'],␣
↪label='Health Sector', linestyle='-', color='green')

# Industrial Sector
plt.plot(data_copy_new_i.index, data_copy_new_i['cumulative_returns_i'],␣
↪label='Industrial Sector', linestyle='-', color='red')

# Equally Weighted Portfolio
plt.plot(portfolio.index, portfolio['cumulative_returns'], label='Equally␣
↪Weighted Portfolio', linestyle='-', color='pink')

# Adding labels and title
plt.xlabel('Date')
plt.ylabel('Cumulative Returns (%)')
plt.title('Comparison of Equity Curves')
plt.legend()
plt.grid(True)
plt.show()
```



As we can see, Portfolio performs better than Industrial Sector but worse than Health Sector.

0.0.12 Q4) Describe a mechanism that may explain why the principle of contrarian opinion may be observed across many financial markets.

As we learnt in class, The Principle of Contrarian Opinion means that financial markets exhibit predictable patterns of behavior in response to the sentiments of speculators. The contrarian pattern suggests that when speculators are excessively optimistic about future prices, the market is likely to experience a subsequent fall, and conversely, when speculators are overly pessimistic, future prices are expected to increase.

One possible mechanism that I can think of that explains why POC may be observed across many financial markets is herding and informational cascades. It begins when a piece of information enters the market, whose true significance may be obscured by its ambiguity in complex financial landscapes. Due to limited information, individual investors struggle with assessing the genuine value of an asset. This creates a cascade effect, making it seem like everyone agrees on the market direction, leading to either overvaluation or undervaluation of assets. Contrarian investors, who do deeper analysis or have different perspectives, take advantage of this trend to make decisions against the popular sentiment, hoping to benefit from the market's overreaction. If their analysis is correct, the market eventually corrects itself, supporting the idea that herding behavior influences financial trends. Another reason for contrarian behavior is the recognition of market inefficiencies and cognitive biases. Markets are not perfect, and sometimes prices don't reflect all available information, causing temporary mispricings of assets. Investors also have biases like anchoring, loss aversion, and overconfidence, which can lead to mistakes. Contrarian investors, who question prevailing market beliefs and rely on fundamental analysis, can identify, and take advantage of these mispricings. By strategically buying undervalued assets or selling overvalued ones, contrarian investors challenge popular sentiments. If their assessments are correct and biases fade away, the market may correct itself, supporting and rewarding the contrarian approach. This two-fold explanation, involving both herding and market inefficiencies, highlights the various factors influencing contrarian behavior in financial markets.

0.0.13 Q5) Higher feds funds rates lead to declines in stock price indexes. True or false, explain.

Generally, it is true. An increase in the federal funds rates typically correlates with declines in stock price indexes. Firstly, as the Federal Reserve raises the federal funds rate, the cost of borrowing for businesses and consumers rises, impacting corporate profits by increasing expenses associated with servicing debts. Secondly, higher interest rates often prompt a shift in investor sentiment, with fixed-income securities becoming more attractive relative to stocks. Consequently, investors may redirect their investments away from stocks, contributing to a decline in stock prices. Additionally, stock prices are influenced by discounting future cash flows, and when interest rates climb, the higher discount rates applied to these future cash flows result in lower present values for future earnings, placing downward pressure on stock prices. Furthermore, while the Federal Reserve may raise interest rates to cool down an overheating economy and control inflation, the subsequent economic slowdown can negatively affect corporate earnings and, in turn, stock prices.