In [1]:

```
# Import necessay library
import pandas as pd
import numpy as np

import functools
from scipy.stats import kurtosis, skew
from itertools import product
import scipy.optimize as sco
import itertools
import random

# matplotlib for plotting
import matplotlib.pyplot as plt

# Suppress warnings from pandas !pip install numpy==1.16.1
import warnings
warnings.filterwarnings('ignore')

# Memory management
import gc

# GARCH model for estimation
from arch import arch_model # FOR garch volatility estimate, IF YOU NOT YET HAVE ARCH MODULE, YOU NEED TO INSTALL IT BY

# set working directory
import os
# os.chdir('C:\\Users\\18183432\\OneDrive - LA TROBE UNIVERSITY\\Python\\trading strategy\\S&P 500 e-Mini futures') # 18
# os.chdir('C:\Users\Owner\OneDrive - LA TROBE UNIVERSITY\Python\trading strategy\S&P 500 e-Mini futures') # Laptop
# os.chdir('C:\\Users\\Phong\\OneDrive - LA TROBE UNIVERSITY\\Python\\trading strategy\\S&P 500 e-Mini futures') # Trust

import os
cwd = os.getcwd()
print(cwd)

from datetime import timedelta
# Set the numbers of columns display
pd.set_option('display.max_columns', 100)


def drawdown (final):
    '''
    function to calculate drawdown of security/portfolio

    Input:
        final: return serires of security/portfolio
    Output:
        drawdown of portfolio
    '''
    cum_rets = (1 + final).cumprod()
    drawdown =  -(1-cum_rets.div(cum_rets.cummax()))

    return drawdown
```

```
C:\Users\Owner\OneDrive - LA TROBE UNIVERSITY\Python\trading strategy\S&P 500 e-Mini futures
```

In [2]:

```
# Read the data of future S&P 500 e-Mini futures (we have total 10 year = 41 contracts)
data = pd.read_csv('SP500_emini_futures_data.csv')
data['Time'] = pd.to_datetime(data['Time'], format=('%d/%m/%Y'))
# Read the SP500 index, choose the start_date for SP500 = 6 years before start_date S&P 500 e-Mini futures (to calculate
sp500 = pd.read_csv('SP500_index.csv', parse_dates =['Date'])[['Date','Adj Close']]
sp500.columns = ['Time', 'SP500_index']
sp500['Time'] = pd.to_datetime(sp500['Time'], format=('%Y/%m/%d'))
start_date = data['Time'][0]- timedelta(365*6)
sp500 = sp500[sp500['Time']>start_date]
```

In [3]:

```
### Calculate GARCH volatility with 3 years window (=3*252 days) based on SP500_index
window = 252*3
forecast_horizon=1
update_regime =5

sp500['ret'] =  sp500['SP500_index'].astype(float).pct_change()*100
ret = np.array(sp500['ret'].dropna())

predictions =[]
realized_vars =[]
price_minus_window = sp500.iloc[window:,:].reset_index().drop(columns = ['index'])
# price_minus_window = price_minus_window.reset_index().drop(columns = ['index'])
for i in range(len(ret)-window+1): # i = 1
    ret1 = np.array(ret[i:i+window])# len(ret1)
    # Record realized variance for comparison with forecast values
    realized_var = np.var(ret1)
    realized_vars.append(realized_var)
    # set up GARCH model
    model = arch_model(ret1, mean='Zero', vol='GARCH', p=1, q=1)#
    # fit model
    model_fit = model.fit()

    # Record forecast values of Variance
    prediction = model_fit.forecast(horizon=forecast_horizon).variance.values[-1]
    predictions.append(np.mean(prediction))

# put the variance and relized variance into data frame
price_minus_window ['forecasted_variance'] = np.array(predictions, float)
price_minus_window ['realized_vars'] = realized_vars
# calculate the median of rolling-3-year window variance
price_minus_window['median_forecast_variance'] = price_minus_window['forecasted_variance'].rolling(window).median()
# eleminate NA values
price_minus_window = price_minus_window.dropna().reset_index().drop(columns = ['index'])
```

```
Iteration:     16,   Func. Count:     95,   Neg. LLF: 933.952174924779
Iteration:     17,   Func. Count:    100,   Neg. LLF: 933.9520941284939
Iteration:     18,   Func. Count:    105,   Neg. LLF: 933.9520927807758
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 933.9520927810576
            Iterations: 18
            Function evaluations: 105
            Gradient evaluations: 18
```

```
Iteration:      1,   Func. Count:      5,   Neg. LLF: 940.4951952715725
Iteration:      2,   Func. Count:     12,   Neg. LLF: 937.7645009954357
Iteration:      3,   Func. Count:     18,   Neg. LLF: 937.2903778359504
Iteration:      4,   Func. Count:     24,   Neg. LLF: 936.5500080796305
Iteration:      5,   Func. Count:     30,   Neg. LLF: 935.917420114929
Iteration:      6,   Func. Count:     36,   Neg. LLF: 935.3868722878653
Iteration:      7,   Func. Count:     42,   Neg. LLF: 934.9298665495774
Iteration:      8,   Func. Count:     48,   Neg. LLF: 934.540332658597
Iteration:      9,   Func. Count:     54,   Neg. LLF: 934.2067141515813
Iteration:     10,   Func. Count:     60,   Neg. LLF: 933.9246659788607
Iteration:     11,   Func. Count:     66,   Neg. LLF: 933.6983624809227
Iteration:     12,   Func. Count:     72,   Neg. LLF: 933.4939560143339
```

In [4]:

```python
### Create a column to indicate volatility regime in SP500_future_data
regime = []
#update_regime = 5 # as the paper said that they only update regime 1 time a week, then update_regime is set by 5 busines
for i in range(update_regime, price_minus_window.shape[0], update_regime): # i = 5, i = 5 +5
    #print(i)
    if price_minus_window['forecasted_variance'][i] > price_minus_window['median_forecast_variance'][i-1]:
        regime += ['low']* update_regime
    else:
        regime += ['high']*update_regime

# Handle the remainder of (number of days / update-regime)
if price_minus_window['forecasted_variance'].iloc[-1] > price_minus_window['median_forecast_variance'].iloc[-2]:
    if (price_minus_window.shape[0] % update_regime) == 0:
        regime += ['low']* update_regime
    else:
        regime += ['low']* (price_minus_window.shape[0] % update_regime)
else:
    if (price_minus_window.shape[0] % update_regime) == 0:
        regime += ['high']* update_regime
    else:
        regime += ['high']* (price_minus_window.shape[0] % update_regime)
   # Put regime back into dataframe
price_minus_window['regime'] = regime

#data = data.drop(df.index[0]).reset_index().drop(columns =['index'])
data = pd.merge(data,price_minus_window[['Time','regime']],  on ='Time', how = 'left')
```

In [5]:

```
bid_ask=pd.read_csv("Bid-Ask Price.csv")
bid_ask.head(10)
#bid_ask.shape[1]
new_header = []
for i in range ((bid_ask.shape[1]-1)//2):
    new_header.append('Bid_{}'.format(str(i)))
    new_header.append('Ask_{}'.format(str(i)))
new_header
bid_ask = bid_ask[5:]
bid_ask['Start Date'] = pd.to_datetime(bid_ask['Start Date'], format=('%d/%m/%Y'))
new_header.insert(0,'Time')
bid_ask.columns = new_header
bid_ask
```

Out[5]:

	Time	Bid_0	Ask_0	Bid_1	Ask_1	Bid_2	Ask_2	Bid_3	Ask_3	Bid_4	Ask_4	Bid_5	Ask_5	Bid_6	Ask_6	Bid_7	Ask_7	Bid_8
5	2008-01-01	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	2008-01-02	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	2008-01-03	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8	2008-01-04	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	2008-01-07	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3124	2019-12-16	1097	1108	1166	1179.5	1116	1116	1128	1128	1243.25	1243.25	1272.25	1272.25	1269.5	1269.5	1210	1210	1217
3125	2019-12-17	1097	1108	1166	1179.5	1116	1116	1128	1128	1243.25	1243.25	1272.25	1272.25	1269.5	1269.5	1210	1210	1217
3126	2019-12-18	1097	1108	1166	1179.5	1116	1116	1128	1128	1243.25	1243.25	1272.25	1272.25	1269.5	1269.5	1210	1210	1217
3127	2019-12-19	1097	1108	1166	1179.5	1116	1116	1128	1128	1243.25	1243.25	1272.25	1272.25	1269.5	1269.5	1210	1210	1217

	Time	Bid_0	Ask_0	Bid_1	Ask_1	Bid_2	Ask_2	Bid_3	Ask_3	Bid_4	Ask_4	Bid_5	Ask_5	Bid_6	Ask_6	Bid_7	Ask_7	Bid_8
3128	2019-12-20	1097	1108	1166	1179.5	1116	1116	1128	1128	1243.25	1243.25	1272.25	1272.25	1269.5	1269.5	1210	1210	1217

3124 rows × 83 columns

In [43]:

```
# Based on the above analysis, I created a function taken into account transaction cost and searching for optimal MA1& MA
def sp500_future_analyzing(data = data, MA1=5, MA2=210, initial_equity1 = 1000000, commission_fee1 = 0.002):
    individual_ret =[]
    valid_contracts = []
    for m in range (data.shape[1]-2):
        #print(m)
        price = data[['Time', str(m), 'regime']]
        price.columns =['Time', 'price', 'regime']
        price['log_ret'] =  price['price'].astype(float).pct_change()*100
        price['low'] = price['price'].rolling(MA1).mean() # 66
        price['high'] = price['price'].rolling(MA2).mean()
        price = price[np.isfinite(price['low'])]
        price = price[np.isfinite(price['high'])].reset_index().drop(columns = ['index'])
        price_minus_window = price
        price_minus_window ['position'] = np.where(price_minus_window['regime'] =='low', np.where(price_minus_window['pr
        price_minus_window ['ret'] = price_minus_window ['position'].shift(1)* price_minus_window ['log_ret']/100
        sell_buy =[]
        for i in range (price_minus_window.shape[0]):
            if (i == 0):
                sell_buy.append('None')

            elif (i == 1):
                if (price_minus_window['position'].iloc[i]== 1):
                    sell_buy.append('buy')

                if (price_minus_window['position'].iloc[i]== -1):
                    sell_buy.append('sell')

            elif (i == price_minus_window.shape[0]-1):
                if (price_minus_window['position'].iloc[i]== 1):
                    sell_buy.append('sell')

                if (price_minus_window['position'].iloc[i]== -1):
                    sell_buy.append('buy')

            else:
                if (price_minus_window['position'].iloc[i-1]== 1) and (price_minus_window['position'].iloc[i]== -1):
                    sell_buy.append('sell')

                elif (price_minus_window['position'].iloc[i-1]== -1) and (price_minus_window['position'].iloc[i]== 1):
                    sell_buy.append('buy')
                else:
                    sell_buy.append('None')
        price_minus_window['sell_buy'] = sell_buy

        price_minus_window = price_minus_window [price_minus_window['ret']!=0]

        # ############# Modeling transaction cost using Bid-Ask Price data #############################################

        # Data for transaction cost
        data_for_transaction_cost = price_minus_window[price_minus_window['sell_buy'].isin(['sell','buy'])].reset_index(
        data_for_transaction_cost = data_for_transaction_cost [['Time', 'sell_buy']]
        transaction_data = data_for_transaction_cost.merge(bid_ask, on='Time', how='left')

        transaction_data1 =transaction_data[['Time','sell_buy', 'Bid_{}'.format(str(m)),'Ask_{}'.format(str(m))]]
        transaction_data1.columns = ['Time','sell_buy','bid','ask']

        # Calculate accumulate equity
        initial_equity = initial_equity1

        commission_fee = [commission_fee1]
        volume =[]
        equity =[]
        for i in range(transaction_data1.shape[0]):

            if (i == 0):
                if (transaction_data1['sell_buy'].iloc[i]== "sell"):
                    equity0 = initial_equity
                    volume0 = initial_equity/float(transaction_data1['bid'].iloc[i])
                else:
                    equity0 = initial_equity
                    volume0 = initial_equity/float(transaction_data1['ask'].iloc[i])
                volume.append(volume0)
                equity.append(equity0)
            elif i == 1:

                if transaction_data1['sell_buy'].iloc[i]== "buy":
                    volume1 = equity[0]/float(transaction_data1['ask'].iloc[i])
                    equity1 = min(volume[0],volume1)*(float(transaction_data1['bid'].iloc[i-1])-float(transaction_data1[
                else:
                    volume1 = equity[0]/float(transaction_data1['bid'].iloc[i])
                    equity1 = min(volume[0],volume1)*(float(transaction_data1['bid'].iloc[i])-float(transaction_data1['a
                volume.append(volume1)
                equity.append(equity1)
            else:
                if equity[i-1] == equity[i-2]:
                    if transaction_data1['sell_buy'].iloc[i]== "buy":
                        volume2 = equity[i-1]/float(transaction_data1['ask'].iloc[i])
                        equity2 = min(volume[i-1],volume2)*(float(transaction_data1['bid'].iloc[i-1])-float(transaction_
                    else:
                        volume2 = equity[i-1]/float(transaction_data1['bid'].iloc[i])
                        equity2 = min(volume[i-1],volume2)*(float(transaction_data1['bid'].iloc[i])-float(transaction_da
                    volume.append(volume2)
                    equity.append(equity2)
                else:
                    equity2 = equity[i-1]
                    if transaction_data1['sell_buy'].iloc[i]== "sell":
                        volume2 = equity[i-1]/float(transaction_data1['bid'].iloc[i])
                    else:
                        volume2 = equity[i-1]/float(transaction_data1['ask'].iloc[i])

                    volume.append(volume2)
                    equity.append(equity2)

        transaction_data1 ['volume'] = volume
        transaction_data1 ['accumulate equity'] = equity
        # Take into aacount the effect of comission fee
        transaction_data1 ['commission_index'] = range(transaction_data1.shape[0])
        transaction_data1 ['commission_fee'] = commission_fee* transaction_data1.shape[0]
        transaction_data1 ['accum_commission_fee'] = transaction_data1 ['commission_index']*transaction_data1 ['commissi
        transaction_data1 ['accum_equity_after_comm'] = transaction_data1 ['accumulate equity'] - transaction_data1 ['ac
        transaction_data1.drop(columns =['commission_index', 'commission_fee'], inplace = True)
        print("Number of transactions in contract ", m, " is ", transaction_data1.shape [0])
        # Put accumulate equity into price_minus_window data frame
        price_minus_window = price_minus_window.merge(transaction_data1[['Time','accum_equity_after_comm']], on = 'Time'
        price_minus_window1 = price_minus_window[['Time', 'accum_equity_after_comm']]
        price_minus_window1.ffill(inplace = True)
        price_minus_window1 ['accum_equity_after_comm'].iloc[0]= price_minus_window1 ['accum_equity_after_comm'].iloc[1]
        price_minus_window['accum_equity_after_comm'] = price_minus_window1['accum_equity_after_comm']
        valid_contracts.append(m)
        individual_ret.append(price_minus_window[['Time','accum_equity_after_comm']])

    ### Collect the returns columns of each contract and merger them tegether based on 'Time' columns
    port_ret =pd.DataFrame(individual_ret[0])
    for i in range(1,len(individual_ret),1): # i = 1
        port_ret = port_ret.merge(pd.DataFrame(individual_ret[i]),on='Time', how='outer')
    ### Create the name for columns in port_ret dataframe
    column1 = valid_contracts.copy()
    column1.insert(0,'Time')
    port_ret.columns = column1
    port_ret.ffill(inplace = True)
    port_ret.fillna(initial_equity, inplace=True)
    port_ret['Total Ret'] = port_ret.sum(axis=1)
    port_ret1 = port_ret.set_index('Time')

    port_ret1['ret'] =  port_ret1['Total Ret'].astype(float).pct_change()*100
    port_ret1

    winning_percentage = len(port_ret1[port_ret1['ret']>0])/len(port_ret1)

    port_ret1['Cum_ret'] = (1+port_ret1['ret']).cumprod() -1
    avg_annualized_return = (1+port_ret1['Cum_ret'].iloc[-1])**(252/port_ret1.shape[0])-1

    print('Average annualized daily return of portfolio is', avg_annualized_return,'%')
    # standard deviation
    annualized_standard_deviation = port_ret1['ret'].std()*np.sqrt(252)
    print('Annualized standard deviation return of portfolio is', annualized_standard_deviation, '%')
    # sharpe ratio
    annualized_sharpe = avg_annualized_return/annualized_standard_deviation
    annualized_sharpe
    print('Annualized_sharpe ratio of portfolio is: ', annualized_sharpe )

    # Maximum draw down
    max_draw_down = max(-drawdown(port_ret1['ret'].dropna()))
    print('Maximum draw down of portfolio is: ', max_draw_down )
    # port_ret.to_csv ('port_ret.csv', index = False, header=True)
    summary_port = pd.DataFrame({'Start':port_ret['Time'].iloc[0] ,'Finish':port_ret['Time'].iloc[-1],'MA1': MA1,'MA2': 
                                 'annualized_Sharpe': annualized_sharpe,'max_draw_down': max_draw_down,'winning_percenta
    return summary_port # port_ret1
```

In [44]:

```
# Run grid_search to find optimal MA1&MA2
from itertools import product
from joblib import Parallel, delayed
import multiprocessing
ma1 = range(10, 70, 5) # ma1 =[5] ma2 = [210]
ma2 = range(21*5,21*15, 21) # ma2 = range(21,21*15, 21)
inputs = list(product(ma1,ma2))
results =[]
for MA1, MA2 in inputs:
    print(MA1, MA2)
    result = sp500_future_analyzing(data = data, MA1=MA1, MA2=MA2, initial_equity1 = 1000000, commission_fee1 = 0.002)
    results.append(result)
for i in range(len(results)):
    if i ==0:
        summary = results[i]
    summary = pd.concat([summary,results[i]], axis =0)
summary.sort_values('annualized_Sharpe', ascending = False)
summary

# Note: the highest achieved Sharpe ratio is 0.56 when we use MA1 = 10 and MA2 = 252
```

```
Number of transactions in contract  40  is  11
Average annualized daily return of portfolio is 0.0752295672487413 %
Annualized standard deviation return of portfolio is 0.32201884148775545 %
Annualized_sharpe ratio of portfolio is:  0.2336185264848916
Maximum draw down of portfolio is:  0.601376299337169
10 252
Number of transactions in contract  0  is  4
Number of transactions in contract  1  is  4
Number of transactions in contract  2  is  11
Number of transactions in contract  3  is  13
Number of transactions in contract  4  is  3
Number of transactions in contract  5  is  2
Number of transactions in contract  6  is  2
Number of transactions in contract  7  is  12
Number of transactions in contract  8  is  14
Number of transactions in contract  9  is  4
Number of transactions in contract  10  is  4
Number of transactions in contract  11  is  4
Number of transactions in contract  12  is  4
Number of transactions in contract  13  is  5
```

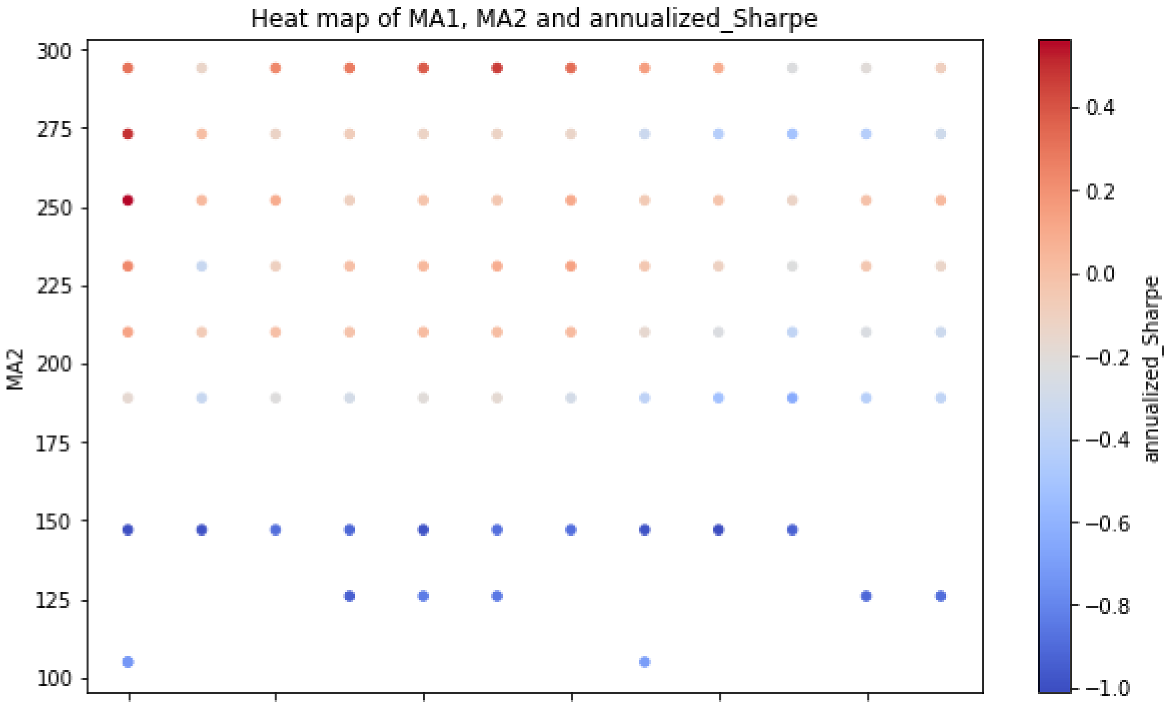
In [45]:
```
summary.sort_values('annualized_Sharpe', ascending = False)
```

Out[45]:

	Start	Finish	MA1	MA2	annualized_return	annualized_sd	annualized_Sharpe	max_draw_down	winning_percentage
0	2009-09-11	2019-12-11	10	252	0.144032	0.254663	0.565579	0.416330	0.027734
0	2009-10-12	2019-12-11	10	273	0.136067	0.273701	0.497138	0.279517	0.025328
0	2009-11-10	2019-12-11	35	294	0.118940	0.252271	0.471479	0.315676	0.026094
0	2009-11-10	2019-12-11	30	294	0.095664	0.249387	0.383598	0.356379	0.026094
0	2009-11-10	2019-12-11	40	294	0.083961	0.256240	0.327665	0.380123	0.022256
...	...	...	...	...	...	...	...	...	...
0	2009-04-17	2019-12-11	60	147	NaN	0.645414	NaN	1.000158	0.019087
0	2009-05-18	2019-12-11	60	168	NaN	0.757131	NaN	1.934011	0.017372
0	2009-02-18	2019-12-11	65	105	NaN	1.331893	NaN	1.000002	0.016206
0	2009-04-17	2019-12-11	65	147	NaN	0.670319	NaN	1.000203	0.018713
0	2009-05-18	2019-12-11	65	168	NaN	0.750850	NaN	1.925471	0.017749

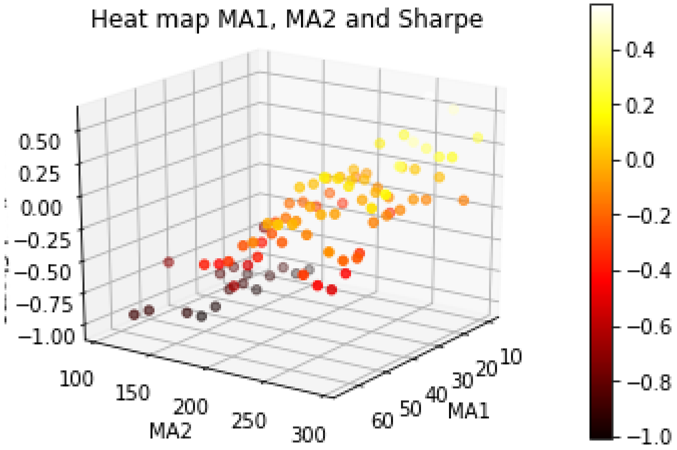
121 rows × 9 columns

```
In [28]: # Plot heat map of Sharpe ratio depending on MA1 and MA2
         %matplotlib inline
         b= summary
         b.plot.scatter(x='MA1', y ='MA2', c='annualized_Sharpe',cmap ='coolwarm', figsize = (10,6), colorbar = True)
         plt.title('Heat map of MA1, MA2 and annualized_Sharpe')
         plt.show()
```

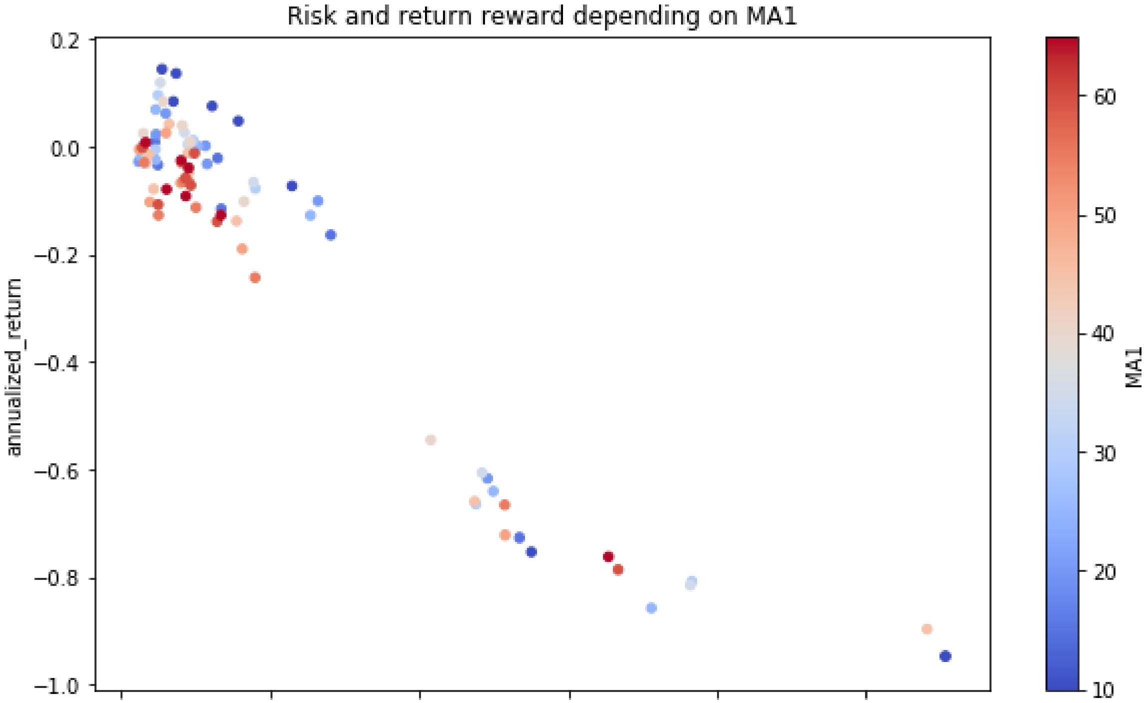


In [29]:

```
# Plot 3D heat map of Sharpe ratio depending on MA1 and MA2
from mpl_toolkits import mplot3d
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('MA1')
ax.set_ylabel('MA2')
ax.set_zlabel('annualized_Sharpe')
ax.set_title('Heat map MA1, MA2 and Sharpe')
ax.view_init(20, 35)
z = b['annualized_Sharpe']
c = b['annualized_Sharpe'] # set color to indicate annualized_Sharpe: the lighter the colour, the higher annualized_Sharp
x = b['MA1']
y = b['MA2']
img = ax.scatter(x, y, z, c=c, cmap=plt.hot())
fig.colorbar(img)
plt.show()
```

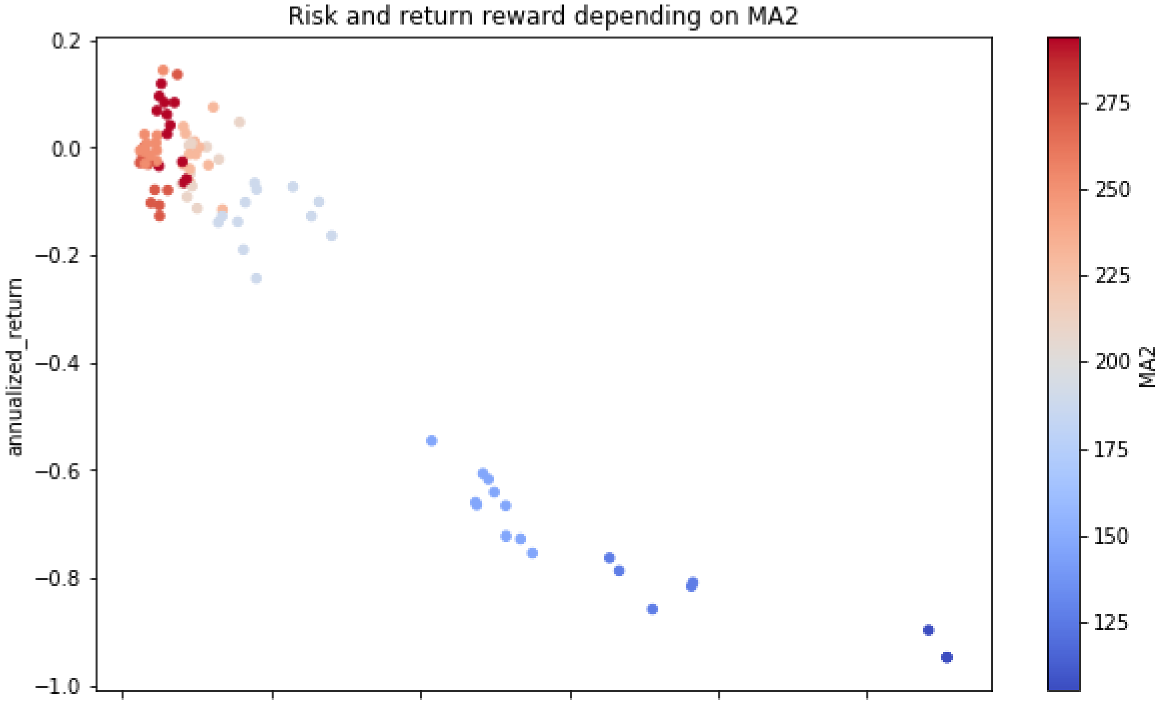


In [30]:
```
# How changes in MA1 affect risk and return

b.plot.scatter(y='annualized_return', x ='annualized_sd', c='MA1',cmap ='coolwarm', figsize = (10,6), colorbar = True)
plt.title('Risk and return reward depending on MA1')
plt.show()
```



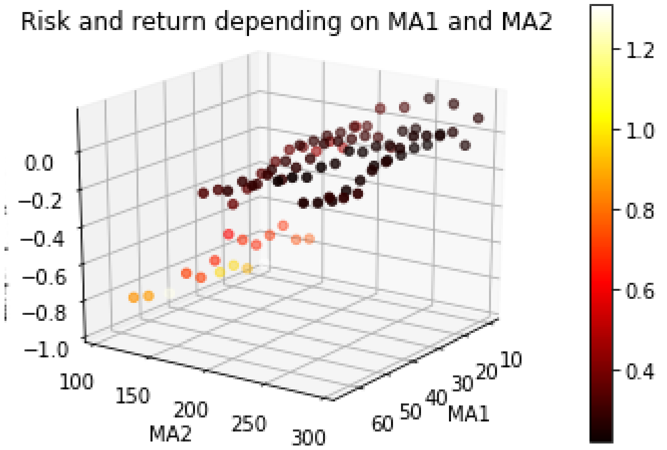
In [31]:

```
# How changes in  MA2 affect risk and return

b.plot.scatter(y='annualized_return', x ='annualized_sd', c='MA2',cmap ='coolwarm', figsize = (10,6), colorbar = True)
plt.title('Risk and return reward depending on MA2')
plt.show()
```



In [32]:

```
########## plot 3D Risk and return reward depending on MA1 and MA2
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('MA1')
ax.set_ylabel('MA2')
ax.set_zlabel('annualized_return')
ax.set_title('Risk and return depending on MA1 and MA2')
ax.view_init(20, 35)
z = b['annualized_return']
c = b['annualized_sd']
x = b['MA1']
y = b['MA2']


img = ax.scatter(x, y, z, c=c, cmap=plt.hot())
fig.colorbar(img)
plt.show()

# Note: I set colour to indicate annualized_sd: the lighter the colour, the higher annualized_sd
```



In [51]:

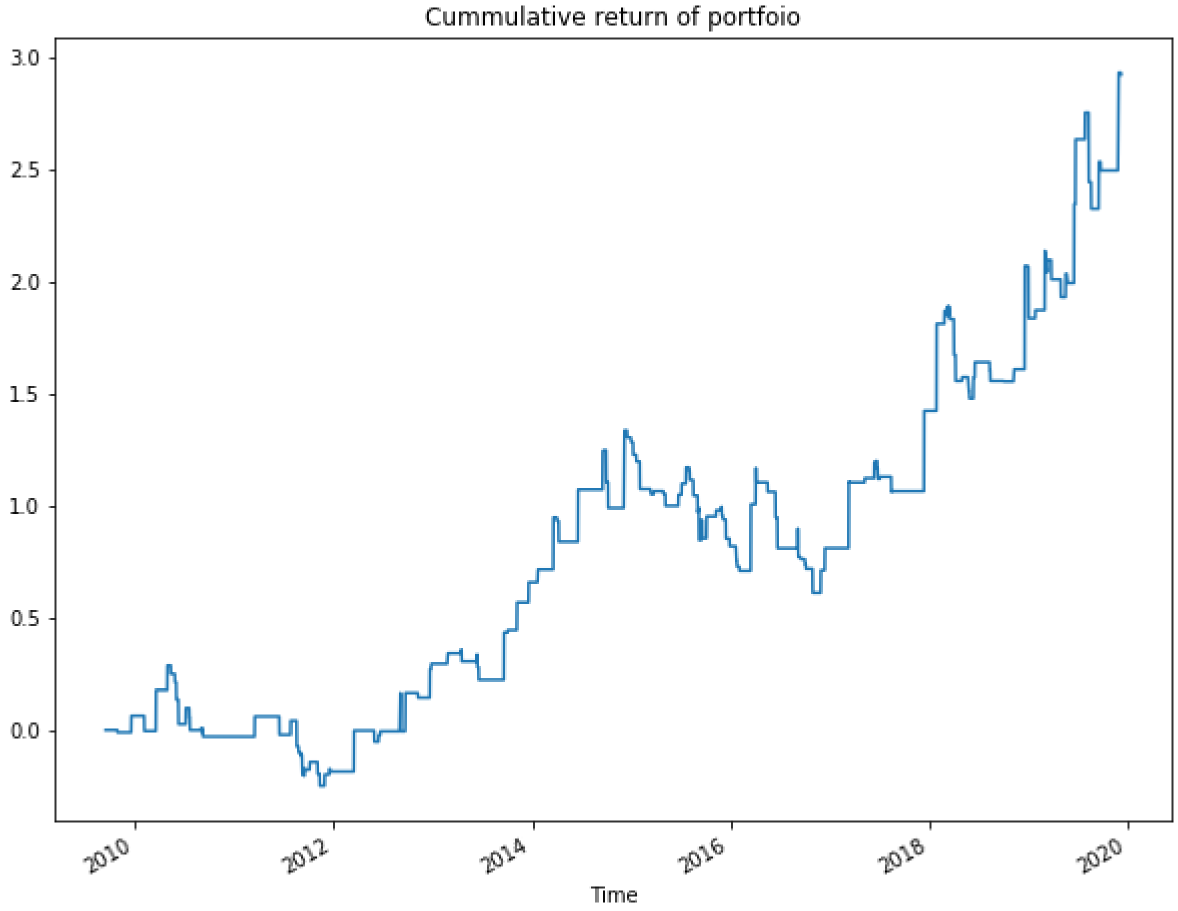
```
# Analyse the performance of max Sharpe case
def get_portfolio_return(data = data, MA1=5, MA2=210, initial_equity1 = 1000000, commission_fee1 = 0.002):
    individual_ret =[]
    valid_contracts = []
    for m in range (data.shape[1]-2):
        #print(m)
        price = data[['Time', str(m), 'regime']]
        price.columns =['Time', 'price', 'regime']
        price['log_ret'] =  price['price'].astype(float).pct_change()*100
        price['low'] = price['price'].rolling(MA1).mean() # 66
        price['high'] = price['price'].rolling(MA2).mean()
        price = price[np.isfinite(price['low'])]
        price = price[np.isfinite(price['high'])].reset_index().drop(columns = ['index'])
        price_minus_window = price
        price_minus_window ['position'] = np.where(price_minus_window['regime'] =='low', np.where(price_minus_window['pr
        price_minus_window ['ret'] = price_minus_window ['position'].shift(1)* price_minus_window ['log_ret']/100
        sell_buy =[]
        for i in range (price_minus_window.shape[0]):
            if (i == 0):
                sell_buy.append('None')

            elif (i == 1):
                if (price_minus_window['position'].iloc[i]== 1):
                    sell_buy.append('buy')

                if (price_minus_window['position'].iloc[i]== -1):
                    sell_buy.append('sell')

            elif (i == price_minus_window.shape[0]-1):
                if (price_minus_window['position'].iloc[i]== 1):
                    sell_buy.append('sell')

                if (price_minus_window['position'].iloc[i]== -1):
                    sell_buy.append('buy')

            else:
                if (price_minus_window['position'].iloc[i-1]== 1) and (price_minus_window['position'].iloc[i]== -1):
                    sell_buy.append('sell')

                elif (price_minus_window['position'].iloc[i-1]== -1) and (price_minus_window['position'].iloc[i]== 1):
                    sell_buy.append('buy')
                else:
                    sell_buy.append('None')
        price_minus_window['sell_buy'] = sell_buy

        price_minus_window = price_minus_window [price_minus_window['ret']!=0]

        # ############# Modeling transaction cost using Bid-Ask Price data #############################################

        # Data for transaction cost
        data_for_transaction_cost = price_minus_window[price_minus_window['sell_buy'].isin(['sell','buy'])].reset_index(
        data_for_transaction_cost = data_for_transaction_cost [['Time', 'sell_buy']]
        transaction_data = data_for_transaction_cost.merge(bid_ask, on='Time', how='left')

        transaction_data1 =transaction_data[['Time','sell_buy', 'Bid_{}'.format(str(m)),'Ask_{}'.format(str(m))]]
        transaction_data1.columns = ['Time','sell_buy','bid','ask']

        # Calculate accumulate equity
        initial_equity = initial_equity1

        commission_fee = [commission_fee1]
        volume =[]
        equity =[]
        for i in range(transaction_data1.shape[0]):

            if (i == 0):
                if (transaction_data1['sell_buy'].iloc[i]== "sell"):
                    equity0 = initial_equity
                    volume0 = initial_equity/float(transaction_data1['bid'].iloc[i])
                else:
                    equity0 = initial_equity
                    volume0 = initial_equity/float(transaction_data1['ask'].iloc[i])
                volume.append(volume0)
                equity.append(equity0)
            elif i == 1:

                if transaction_data1['sell_buy'].iloc[i]== "buy":
                    volume1 = equity[0]/float(transaction_data1['ask'].iloc[i])
                    equity1 = min(volume[0],volume1)*(float(transaction_data1['bid'].iloc[i-1])-float(transaction_data1[
                else:
                    volume1 = equity[0]/float(transaction_data1['bid'].iloc[i])
                    equity1 = min(volume[0],volume1)*(float(transaction_data1['bid'].iloc[i])-float(transaction_data1['a
                volume.append(volume1)
                equity.append(equity1)
            else:
                if equity[i-1] == equity[i-2]:
                    if transaction_data1['sell_buy'].iloc[i]== "buy":
                        volume2 = equity[i-1]/float(transaction_data1['ask'].iloc[i])
                        equity2 = min(volume[i-1],volume2)*(float(transaction_data1['bid'].iloc[i-1])-float(transaction_
                    else:
                        volume2 = equity[i-1]/float(transaction_data1['bid'].iloc[i])
                        equity2 = min(volume[i-1],volume2)*(float(transaction_data1['bid'].iloc[i])-float(transaction_da
                    volume.append(volume2)
                    equity.append(equity2)
                else:
                    equity2 = equity[i-1]
                    if transaction_data1['sell_buy'].iloc[i]== "sell":
                        volume2 = equity[i-1]/float(transaction_data1['bid'].iloc[i])
                    else:
                        volume2 = equity[i-1]/float(transaction_data1['ask'].iloc[i])

                    volume.append(volume2)
                    equity.append(equity2)

        transaction_data1 ['volume'] = volume
        transaction_data1 ['accumulate equity'] = equity
        # Take into aacount the effect of comission fee
        transaction_data1 ['commission_index'] = range(transaction_data1.shape[0])
        transaction_data1 ['commission_fee'] = commission_fee* transaction_data1.shape[0]
        transaction_data1 ['accum_commission_fee'] = transaction_data1 ['commission_index']*transaction_data1 ['commissi
        transaction_data1 ['accum_equity_after_comm'] = transaction_data1 ['accumulate equity'] - transaction_data1 ['ac
        transaction_data1.drop(columns =['commission_index', 'commission_fee'], inplace = True)
        print("Number of transactions in contract ", m, " is ", transaction_data1.shape [0])
        # Put accumulate equity into price_minus_window data frame
        price_minus_window = price_minus_window.merge(transaction_data1[['Time','accum_equity_after_comm']], on = 'Time'
        price_minus_window1 = price_minus_window[['Time', 'accum_equity_after_comm']]
        price_minus_window1.ffill(inplace = True)
        price_minus_window1 ['accum_equity_after_comm'].iloc[0]= price_minus_window1 ['accum_equity_after_comm'].iloc[1]
        price_minus_window['accum_equity_after_comm'] = price_minus_window1['accum_equity_after_comm']
        valid_contracts.append(m)
        individual_ret.append(price_minus_window[['Time','accum_equity_after_comm']])

    ### Collect the returns columns of each contract and merger them tegether based on 'Time' columns
    port_ret =pd.DataFrame(individual_ret[0])
    for i in range(1,len(individual_ret),1): # i = 1
        port_ret = port_ret.merge(pd.DataFrame(individual_ret[i]),on='Time', how='outer')
    ### Create the name for columns in port_ret dataframe
    column1 = valid_contracts.copy()
    column1.insert(0,'Time')
    port_ret.columns = column1
    port_ret.ffill(inplace = True)
    port_ret.fillna(initial_equity, inplace=True)
    port_ret['Total Ret'] = port_ret.sum(axis=1)
    port_ret1 = port_ret.set_index('Time')

    port_ret1['ret'] =  port_ret1['Total Ret'].astype(float).pct_change()*100
    port_ret1

    winning_percentage = len(port_ret1[port_ret1['ret']>0])/len(port_ret1)

    port_ret1['Cum_ret'] = (1+port_ret1['ret']).cumprod() -1
    avg_annualized_return = (1+port_ret1['Cum_ret'].iloc[-1])**(252/port_ret1.shape[0])-1

    print('Average annualized daily return of portfolio is', avg_annualized_return,'%')
    # standard deviation
    annualized_standard_deviation = port_ret1['ret'].std()*np.sqrt(252)
    print('Annualized standard deviation return of portfolio is', annualized_standard_deviation, '%')
    # sharpe ratio
    annualized_sharpe = avg_annualized_return/annualized_standard_deviation
    annualized_sharpe
    print('Annualized_sharpe ratio of portfolio is: ', annualized_sharpe )

    # Maximum draw down
    max_draw_down = max(-drawdown(port_ret1['ret'].dropna()))
    print('Maximum draw down of portfolio is: ', max_draw_down )
    # port_ret.to_csv ('port_ret.csv', index = False, header=True)
    summary_port = pd.DataFrame({'Start':port_ret['Time'].iloc[0] ,'Finish':port_ret['Time'].iloc[-1],'MA1': MA1,'MA2': 
                                 'annualized_Sharpe': annualized_sharpe,'max_draw_down': max_draw_down,'winning_percenta
    return port_ret1

port_ret1 = get_portfolio_return(data = data, MA1=10, MA2=252, initial_equity1 = 1000000, commission_fee1 = 0.002)
```

```
Number of transactions in contract  0  is  4
Number of transactions in contract  1  is  4
Number of transactions in contract  2  is  11
Number of transactions in contract  3  is  13
Number of transactions in contract  4  is  3
Number of transactions in contract  5  is  2
Number of transactions in contract  6  is  2
Number of transactions in contract  7  is  12
Number of transactions in contract  8  is  14
Number of transactions in contract  9  is  4
Number of transactions in contract  10  is  4
Number of transactions in contract  11  is  4
Number of transactions in contract  12  is  4
Number of transactions in contract  13  is  5
Number of transactions in contract  14  is  11
Number of transactions in contract  15  is  8
Number of transactions in contract  16  is  6
Number of transactions in contract  17  is  4
Number of transactions in contract  18  is  6
Number of transactions in contract  19  is  2
Number of transactions in contract  20  is  10
Number of transactions in contract  21  is  13
Number of transactions in contract  22  is  9
Number of transactions in contract  23  is  18
Number of transactions in contract  24  is  19
Number of transactions in contract  25  is  10
Number of transactions in contract  26  is  8
Number of transactions in contract  27  is  7
Number of transactions in contract  28  is  12
Number of transactions in contract  29  is  4
Number of transactions in contract  30  is  6
Number of transactions in contract  31  is  13
Number of transactions in contract  32  is  2
Number of transactions in contract  33  is  8
Number of transactions in contract  34  is  16
Number of transactions in contract  35  is  7
Number of transactions in contract  36  is  6
Number of transactions in contract  37  is  10
Number of transactions in contract  38  is  12
Number of transactions in contract  39  is  10
Number of transactions in contract  40  is  6
Average annualized daily return of portfolio is 0.14403210222481588 %
Annualized standard deviation return of portfolio is 0.2546629488250348 %
```

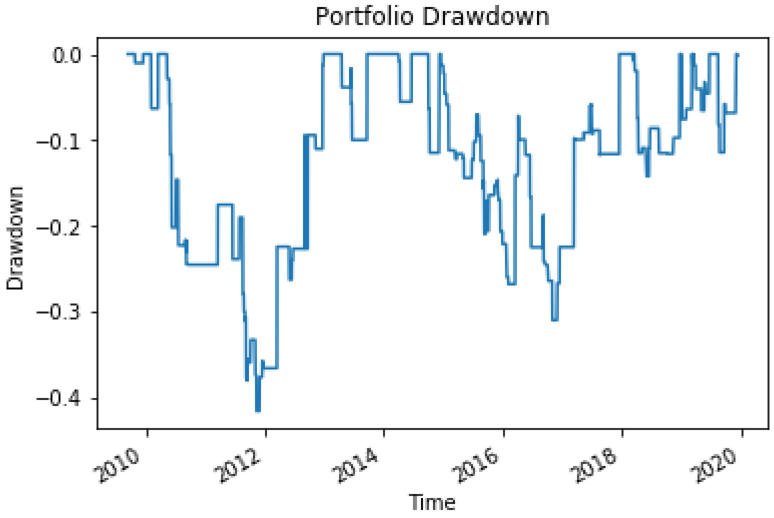
```
Annualized_sharpe ratio of portfolio is:  0.565579338845136
Maximum draw down of portfolio is:  0.41632980701273037
```

In [47]:

```
# Show the cummulative return
port_ret1['Cum_ret'].plot(figsize=(10,8))
plt.title('Cummulative return of portfoio')
plt.show()
```



```
In [48]: # Exhibit 8 : Portfolio drawdown overtime.

         import matplotlib.pyplot as plt
         drawdown(port_ret1['ret']).plot()
         plt.title('Portfolio Drawdown')
         plt.xlabel('Time')
         plt.ylabel('Drawdown')
         plt.show()
```



In [49]:

```
# Exhibit_6: Out of sample performance summary
port_return =pd.DataFrame(port_ret1['ret'])
port_return.index = pd.to_datetime(port_return.index)
port_return['Year'] = port_return.index.year
ret1 = []
vol1 =[]
sharpe1=[]
sortino1=[]
max_drawdown1=[]
year1 = []
for year in list(np.unique(port_return['Year'])): # year = 2014
    return_year = port_return[port_return['Year']==year]
    return_year['Cum_ret'] = (1+return_year['ret']).cumprod() -1
    a_ret = (1+return_year['Cum_ret'].iloc[-1])**(252/return_year['Cum_ret'].shape[0])-1 # annualize
    ret1.append(a_ret)
    a_vol = return_year['ret'].std()*252**0.5
    vol1.append(a_vol) #252**0.5*sharpe
    sharpe1.append(a_ret/a_vol)
    max_drawdown1.append(max(-drawdown(return_year['ret'])))
    year1.append(year)
    # Need a short rate to calculate the sortino
exhibit_6 = pd.DataFrame(list(zip(year1,ret1, vol1,sharpe1, max_drawdown1)),
                         columns =['Year','Annualize_return','Annualize_volatility','Annualize_sharpe', 'MaxDrawdown'])
exhibit_6
```

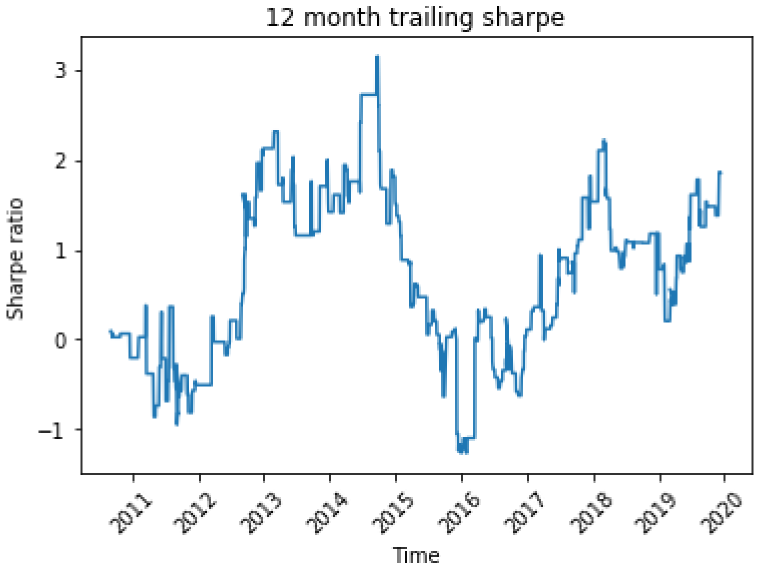
Out[49]:

	Year	Annualize_return	Annualize_volatility	Annualize_sharpe	MaxDrawdown
0	2009	0.219752	0.136759	1.606855	NaN
1	2010	-0.087801	0.269921	-0.325285	0.245721
2	2011	-0.163455	0.261814	-0.624317	0.292011
3	2012	0.603147	0.312354	1.930971	0.049694
4	2013	0.282557	0.218975	1.290363	0.099840
5	2014	0.399546	0.289733	1.379014	0.114873
6	2015	-0.212352	0.160407	-1.323833	0.203402
7	2016	-0.004651	0.264983	-0.017551	0.256936
8	2017	0.343559	0.246524	1.393614	0.062849

	Year	Annualize_return	Annualize_volatility	Annualize_sharpe	MaxDrawdown
9	2018	0.270700	0.260584	1.038822	0.142350
10	2019	0.294662	0.261500	1.126813	0.114413

In [50]:

```
# Exhibit 7: 12 month trailing sharpe
port_return['12M_Sharpe'] = (((1+port_return['ret'].rolling(252).mean())**252)-1)/(port_return['ret'].rolling(252).std()

import matplotlib.pyplot as plt
plt.plot(port_return['12M_Sharpe'])
plt.xticks(rotation=45)
plt.xlabel('Time')
plt.ylabel('Sharpe ratio')
plt.title('12 month trailing sharpe')
plt.show()
```



In [ ]: