Leverage II: Rebalancing

In [1]:
```
# Working with data:
import numpy  as np                                    # For scientific computing
import pandas as pd                                    # Working with tables.

# Downloading files:
import requests, zipfile, io                                 # To access websites

# Specific data providers:
from tiingo import TiingoClient                        # Stock prices.
import quandl                                          # Economic data, futures p

# API keys:
tiingo = TiingoClient({'api_key':'XXXX'})
quandl.ApiConfig.api_key = 'YYYY'

# Plotting:
import matplotlib.pyplot as plt                         # Basic plot library.
plt.style.use('ggplot')                                 # Make plots look nice
```

Get prices:

In [2]:
```
PRICE       = tiingo.get_dataframe(['SPY'],'1900-01-01', metric_name='adjClose')
PRICE.index = pd.to_datetime(PRICE.index).tz_convert(None)
```

In [3]:
```
RET = PRICE.pct_change()
RET[-3:]
```

Out[3]:

	SPY
2021-03-29	-0.000505
2021-03-30	-0.002653
2021-03-31	0.004053

In [4]:
```
fedfunds = quandl.get(['FRED/FEDFUNDS']).rename(columns={'FRED/FEDFUNDS - Value'
fedfunds[-3:]
```

Out[4]:

	Fedfunds
Date	
2020-12-01	0.000004
2021-01-01	0.000004
2021-02-01	0.000003

Assume margin rate equals fed funds rate + 100 basis points (1%):

In [5]:
```
RET = RET.join(fedfunds, how='outer')
```

```
RET['Fedfunds']   = RET.Fedfunds.ffill()
RET['MarginRate'] = RET.Fedfunds + 0.01/252
RET = RET.dropna()
RET
```

Out[5]:

	SPY	Fedfunds	MarginRate
1993-02-01	0.007112	0.000120	0.000160
1993-02-02	0.002118	0.000120	0.000160
1993-02-03	0.010572	0.000120	0.000160
1993-02-04	0.004184	0.000120	0.000160
1993-02-05	-0.000696	0.000120	0.000160
...	...	...	...
2021-03-25	0.005626	0.000003	0.000043
2021-03-26	0.016115	0.000003	0.000043
2021-03-29	-0.000505	0.000003	0.000043
2021-03-30	-0.002653	0.000003	0.000043
2021-03-31	0.004053	0.000003	0.000043

7093 rows × 3 columns

In [7]:

```
weights = pd.Series({'SPY':1.5, 'MarginRate':-0.5})
weights
```

Out[7]:

```
SPY           1.5
MarginRate   -0.5
dtype: float64
```

Compound weighted average returns:

In [8]:

```
RET.multiply(weights).sum('columns').add(1).cumprod()  # rebalance every day
```

Out[8]:

```
1993-02-01     1.010589
1993-02-02     1.013718
1993-02-03     1.029712
1993-02-04     1.036092
1993-02-05     1.034929
                 ...
2021-03-25    24.536645
2021-03-26    25.129230
2021-03-29    25.109653
2021-03-30    25.009192
2021-03-31    25.160715
Length: 7093, dtype: float64
```

Weighted average of compound returns:

In [9]:

```
RET.add(1).cumprod()  # $1 initial investment in each asset and never rebalance
```

Out[9]:

	SPY	Fedfunds	MarginRate

	SPY	Fedfunds	MarginRate
1993-02-01	1.007112	1.000120	1.000160
1993-02-02	1.009245	1.000240	1.000320
1993-02-03	1.019915	1.000361	1.000480
1993-02-04	1.024182	1.000481	1.000640
1993-02-05	1.023470	1.000601	1.000800
...	...	...	...
2021-03-25	15.059675	2.013265	2.667213
2021-03-26	15.302362	2.013272	2.667328
2021-03-29	15.294633	2.013278	2.667442
2021-03-30	15.254056	2.013285	2.667556
2021-03-31	15.315887	2.013291	2.667670

7093 rows × 3 columns

In [10]:
```
RET.add(1).cumprod().multiply(weights)  # initial investments: $1.5 in SPY and $
```

Out[10]:

	Fedfunds	MarginRate	SPY
1993-02-01	NaN	-0.500080	1.510669
1993-02-02	NaN	-0.500160	1.513867
1993-02-03	NaN	-0.500240	1.529872
1993-02-04	NaN	-0.500320	1.536273
1993-02-05	NaN	-0.500400	1.535205
...	...	...	...
2021-03-25	NaN	-1.333607	22.589513
2021-03-26	NaN	-1.333664	22.953542
2021-03-29	NaN	-1.333721	22.941949
2021-03-30	NaN	-1.333778	22.881084
2021-03-31	NaN	-1.333835	22.973831

7093 rows × 3 columns

In [11]:
```
RET.add(1).cumprod().multiply(weights).sum('columns') # Total value of the portf
```

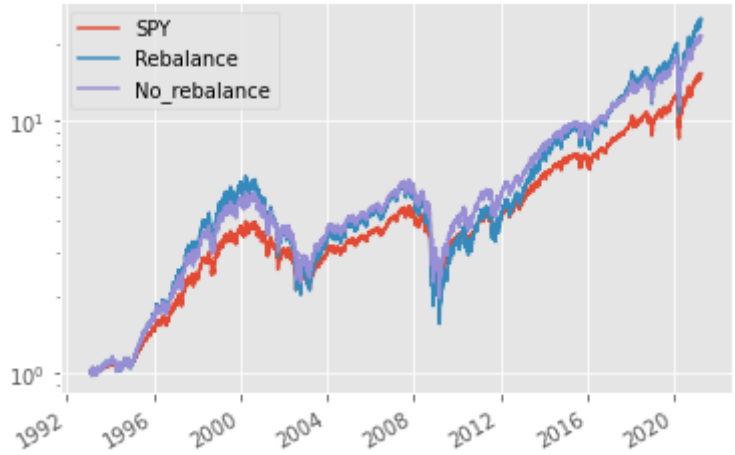
Out[11]:
```
1993-02-01     1.010589
1993-02-02     1.013707
1993-02-03     1.029632
1993-02-04     1.035953
1993-02-05     1.034805
                 ...
2021-03-25    21.255907
```

```
2021-03-26    21.619879
2021-03-29    21.608228
2021-03-30    21.547306
2021-03-31    21.639995
Length: 7093, dtype: float64
```

Compare these two strategies:

In [12]:
```
t = pd.DataFrame()
t['SPY']          = RET.SPY.add(1).cumprod()
t['Rebalance']    = RET[['SPY','MarginRate']].multiply(weights).sum('columns').a
t['No_rebalance'] = RET[['SPY','MarginRate']].add(1).cumprod().multiply(weights)

t.plot(logy=True)
```

Out[12]: <AxesSubplot:>



Portfolio positions (dollars invested) if we do not rebalance:

In [13]:
```
positions = RET[['SPY','MarginRate']].add(1).cumprod().multiply(weights)
positions
```

Out[13]:

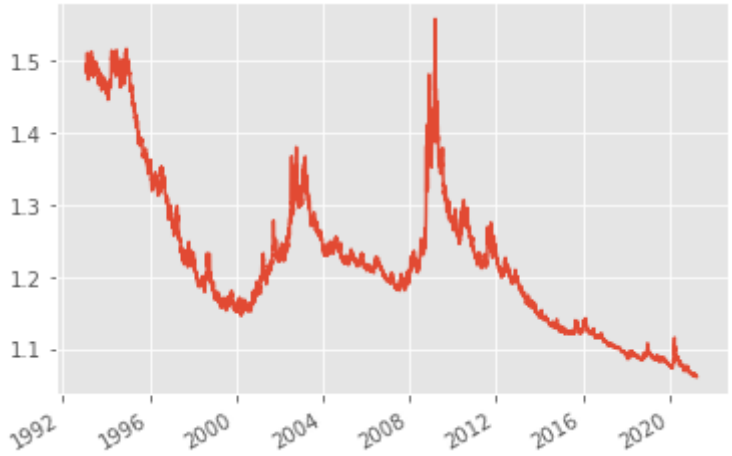
	SPY	MarginRate
1993-02-01	1.510669	-0.500080
1993-02-02	1.513867	-0.500160
1993-02-03	1.529872	-0.500240
1993-02-04	1.536273	-0.500320
1993-02-05	1.535205	-0.500400
...	...	...
2021-03-25	22.589513	-1.333607
2021-03-26	22.953542	-1.333664
2021-03-29	22.941949	-1.333721
2021-03-30	22.881084	-1.333778
2021-03-31	22.973831	-1.333835

7093 rows × 2 columns

Leverage ratio:

In [14]:
```
(positions.SPY / positions.sum('columns')).plot()
```

Out[14]: <AxesSubplot:>



High water mark:

In [15]:
```
hwm = t.cummax()   # cummax: maximum value from 1st row to current row (date)
hwm
```

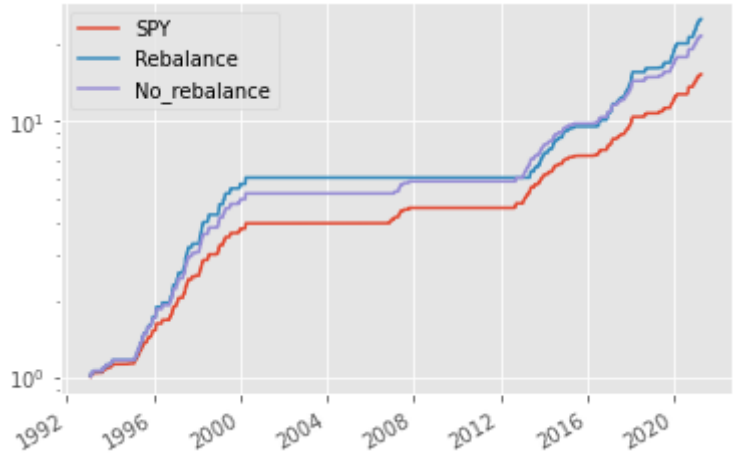
Out[15]:

	SPY	Rebalance	No_rebalance
1993-02-01	1.007112	1.010589	1.010589
1993-02-02	1.009245	1.013718	1.013707
1993-02-03	1.019915	1.029712	1.029632
1993-02-04	1.024182	1.036092	1.035953
1993-02-05	1.024182	1.036092	1.035953
...	...	...	...
2021-03-25	15.301625	25.137373	21.619174
2021-03-26	15.302362	25.137373	21.619879
2021-03-29	15.302362	25.137373	21.619879
2021-03-30	15.302362	25.137373	21.619879
2021-03-31	15.315887	25.160715	21.639995

7093 rows × 3 columns

In [16]:
```
hwm.plot(logy=True)
```

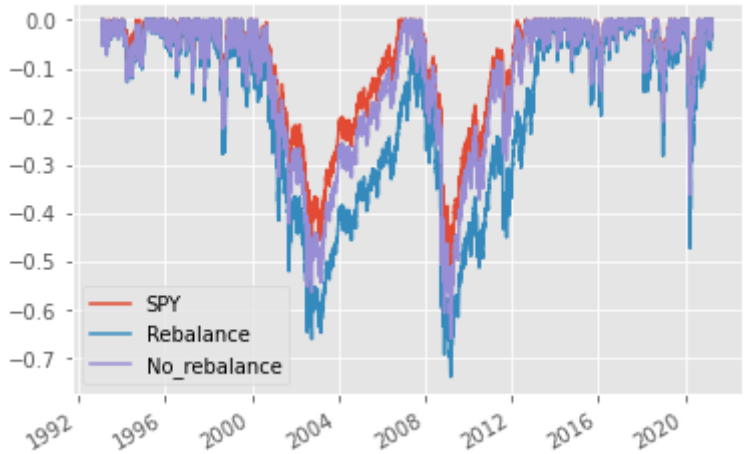
Out[16]: <AxesSubplot:>



Portfolio drawdown:

In [17]:
```
drawdown = t/hwm - 1.0    # % portfolio loss relative to most recent peak (high w
drawdown.plot()
```

Out[17]: <AxesSubplot:>

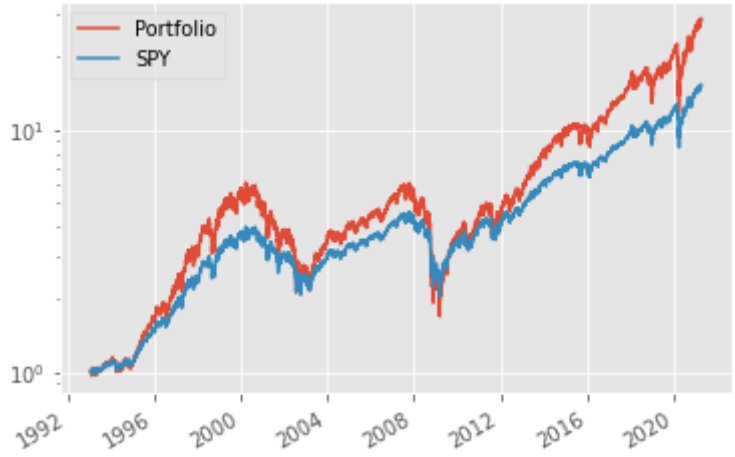


With rebalance loop:

In [18]:
```
def get_rebalance_dates(frequency):
    group = getattr(PRICE.index, frequency)
    return PRICE[:1].index.union(PRICE.groupby([PRICE.index.year, group]).tail(1



def run_backtest(frequency):

    weights = pd.Series({'SPY':1.5, 'MarginRate':-0.5})

    rebalance_dates = get_rebalance_dates(frequency)

    portfolio_value = pd.Series(1,                          index=[rebalance_dates
    leverage        = pd.Series(weights.SPY,                index=[rebalance_dates
    trades          = pd.DataFrame(columns=weights.index, index=[rebalance_dates
    previous_positions = weights

    for i in range(len(rebalance_dates)-1):
        start_date = rebalance_dates[i]
```

```
        end_date    = rebalance_dates[i+1]

        cum_ret = RET[start_date:end_date][1:].add(1).cumprod()

        new_positions = portfolio_value.iloc[-1] * weights

        start_to_end_positions = new_positions  * cum_ret

        portfolio_value = portfolio_value.append(start_to_end_positions.sum('col
        leverage = leverage.append(start_to_end_positions.SPY / start_to_end_pos

        trades.loc[start_date] = new_positions - previous_positions
        previous_positions     = start_to_end_positions.iloc[-1]      # Previous

    return portfolio_value, leverage, trades


portfolio_value, leverage, trades = run_backtest('month')

portfolio_value.to_frame('Portfolio').join(RET.SPY.add(1).cumprod()).plot(logy=T
```

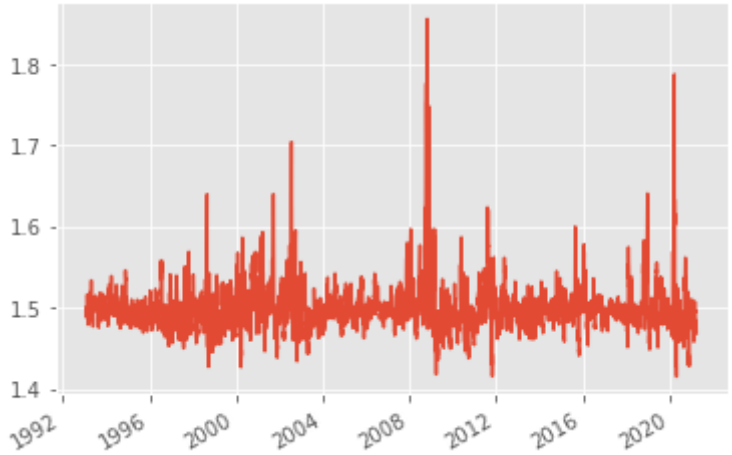
Out[18]: <AxesSubplot:>



Leverage:

In [19]:
```
leverage.plot()  # Leverage if we rebalance once per month to target weights 150
```

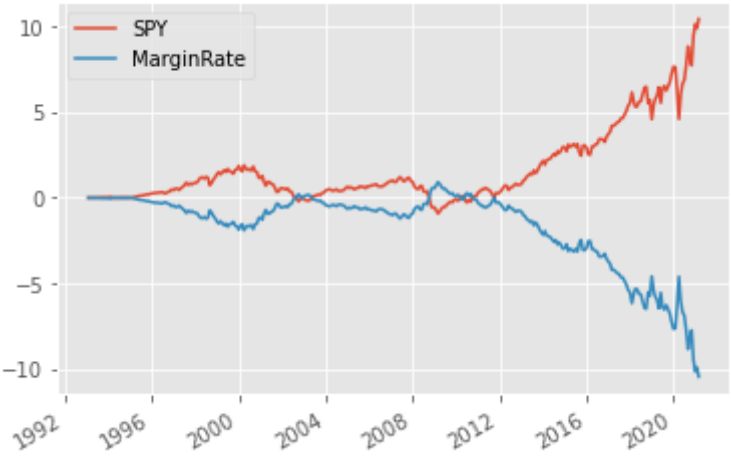
Out[19]: <AxesSubplot:>



Cumulative trades:

In [20]:
```
trades.cumsum().plot()      # Total dollars moved between cash and stocks
```

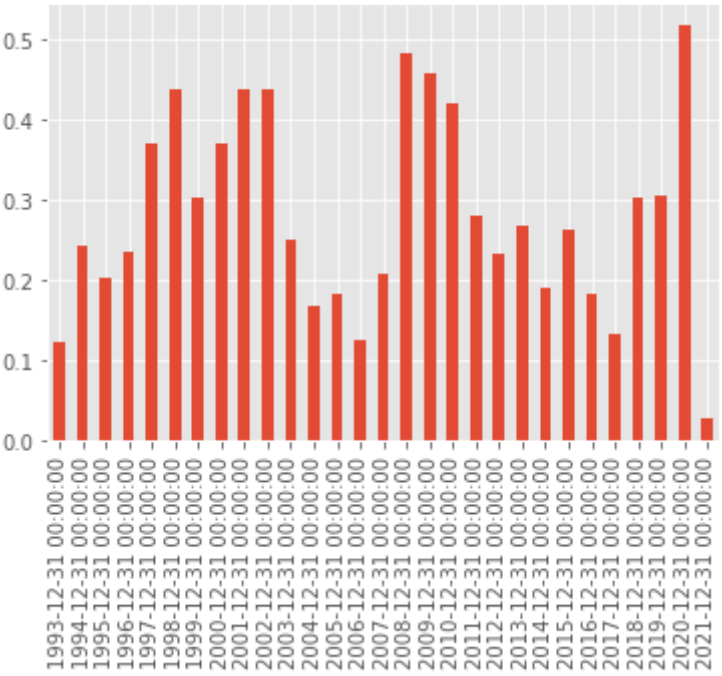
Out[20]: <AxesSubplot:>



Annual turnover ratio:

In [21]:
```
turnover = trades.abs().sum('columns').div(2)
turnover.resample('A').sum().div( portfolio_value.resample('A').mean() ).plot.ba
```

Out[21]: <AxesSubplot:>



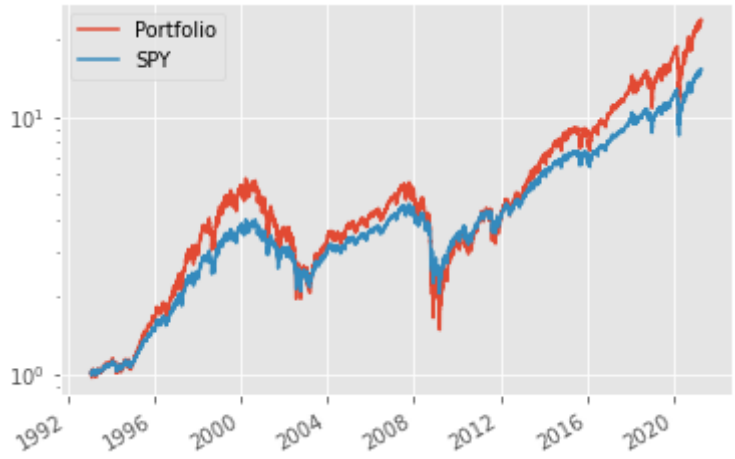
Rebalance only when leverage too far away from target weights:

- leverage > 1.7: rebalance back to 1.5
- leverage < 1.3: rebalance back to 1.5
- else: do nothing

In [23]:
```

```

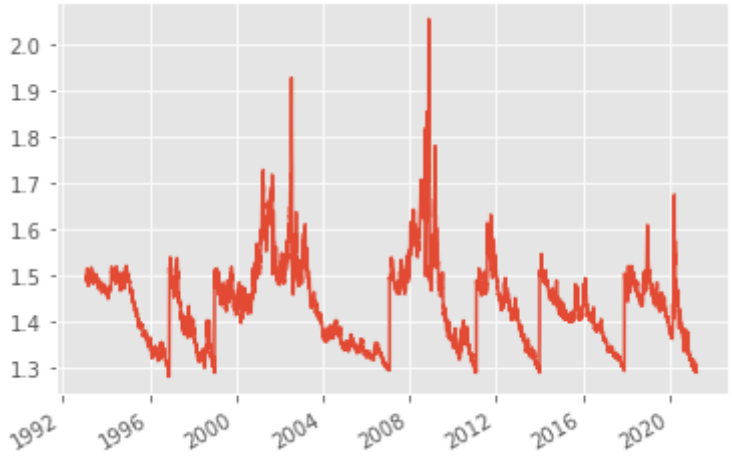
```
def run_backtest(frequency):

    weights = pd.Series({'SPY':1.5, 'MarginRate':-0.5})

    rebalance_dates = get_rebalance_dates(frequency) # PRICE.index

    portfolio_value = pd.Series(1,                          index=[rebalance_dates
    leverage        = pd.Series(weights.SPY,                index=[rebalance_dates
    trades          = pd.DataFrame(columns=weights.index, index=[rebalance_dates
    previous_positions = weights

    for i in range(len(rebalance_dates)-1):
        start_date = rebalance_dates[i]
        end_date   = rebalance_dates[i+1]

        cum_ret = RET[start_date:end_date][1:].add(1).cumprod()

        if((leverage[-1]<weights.SPY-0.2) | (leverage[-1]>weights.SPY+0.2)):
            new_positions = portfolio_value.iloc[-1] * weights
        else:
            new_positions = previous_positions

        start_to_end_positions = new_positions  * cum_ret

        portfolio_value = portfolio_value.append(start_to_end_positions.sum('col
        leverage = leverage.append(start_to_end_positions.SPY / start_to_end_pos

        trades.loc[start_date]  = new_positions - previous_positions
        previous_positions      = start_to_end_positions.iloc[-1]      # Previou

    return portfolio_value, leverage, trades


portfolio_value, leverage, trades = run_backtest('month')


portfolio_value.to_frame('Portfolio').join(RET.SPY.add(1).cumprod()).plot(logy=T
```

Out[23]: <AxesSubplot:>



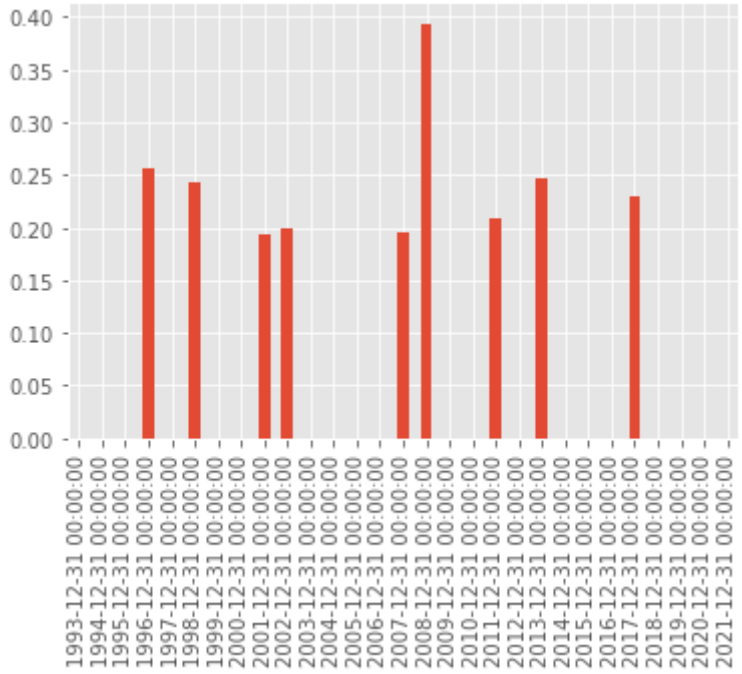
In [24]:
```
leverage.plot()
```

Out[24]: <AxesSubplot:>



In [25]:
```
turnover = trades.abs().sum('columns').div(2)
turnover.resample('A').sum().div( portfolio_value.resample('A').mean() ).plot.ba
```

Out[25]: <AxesSubplot:>



In [ ]:

In [ ]: