Optimal Portfolio Weights III: Leverage

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
```

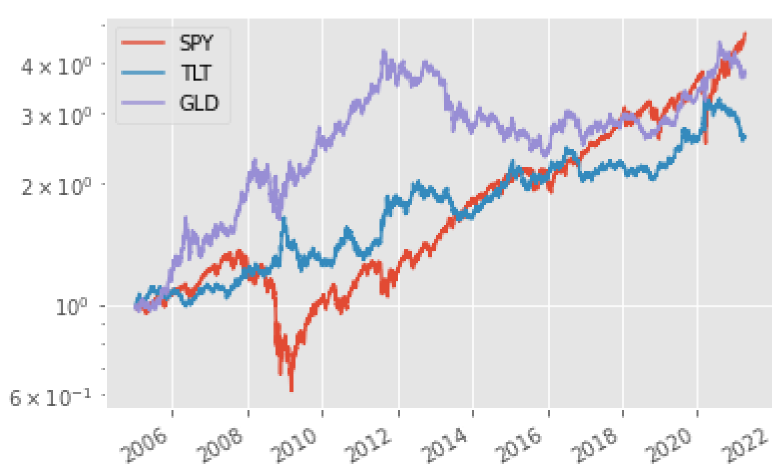
Get data

Get ETF prices and returns (GLD: Gold ETF, TLT: 20+ year treasuries:

In [2]:

```
# start in 2005 since GLD not available earlier
PRICE = tiingo.get_dataframe(['SPY','GLD','TLT'], '2005-1-1', metric_name='adjCl
PRICE.index = pd.to_datetime(PRICE.index).tz_convert(None)

RATES = quandl.get(['FRED/FEDFUNDS','FRED/DGS1']) / 100
RATES.columns = ['FedFunds','Treasury_1']

RET = PRICE.pct_change()
RET = RET.join(RATES.FedFunds.rename('MarginLoan'), how='outer')
RET['MarginLoan'] = RET.MarginLoan.ffill()/252 + 0.01/252          # Assume mar
RET = RET.dropna(subset=['SPY'])

RET[['SPY','TLT','GLD']].add(1).cumprod().plot(logy=True)
```

Out[2]: <AxesSubplot:>



Backtesting the minimum volatility strategy

Basic backtest loop:

In [3]:

```
def get_rebalance_dates(frequency, start_date):
    price = PRICE[PRICE.index>start_date]              # Rebalance dates start a
    group = getattr(price.index, frequency)
    return price[:1].index.union(price.groupby([price.index.year, group]).tail(1




def run_backtest(frequency, backtest_start='1900-1-1'):   # backtest_start: opti

    rebalance_dates = get_rebalance_dates(frequency, backtest_start)

    portfolio_value = pd.Series(1,                            index=[rebalance_dates[0
    weights          = pd.DataFrame(columns=RET.columns, index=[rebalance_dates[0
    trades           = pd.DataFrame(columns=RET.columns, index=[rebalance_dates[0

    previous_positions = weights.iloc[0]

    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()

        start_weights = select_weights(start_date)

        new_positions = portfolio_value.iloc[-1] * start_weights

        start_to_end_positions = new_positions  * cum_ret
        start_to_end_value     = start_to_end_positions.sum('columns')

        portfolio_value = portfolio_value.append(start_to_end_value)

        weights = weights.append(start_to_end_positions.div(start_to_end_value,'

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

    return portfolio_value.pct_change(), weights, trades
```

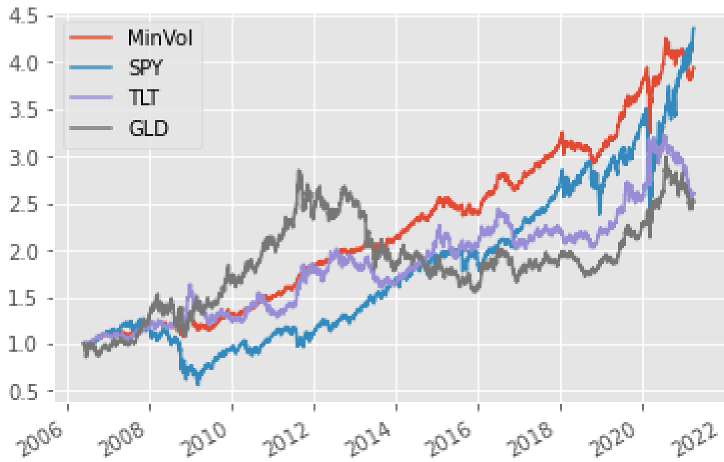
Run the backtest:

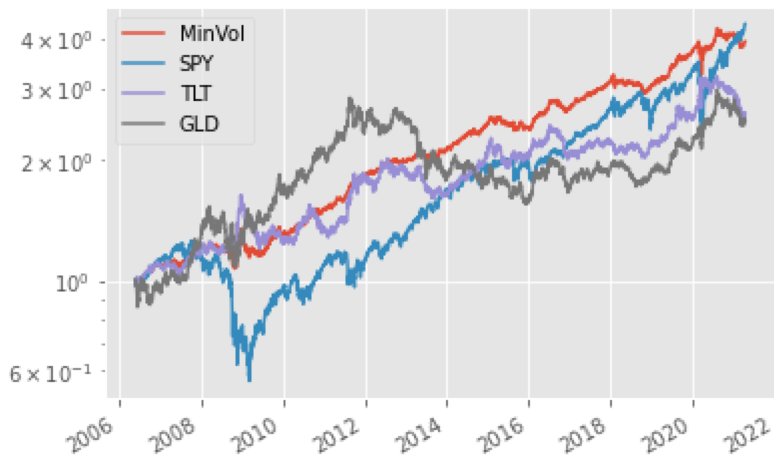
In [4]:

```
def select_weights(date):
        cov     = RET[['SPY','TLT','GLD']][:date][-100:].cov() * 252
        cov_inv = pd.DataFrame(np.linalg.inv(cov), columns=cov.columns, index=co

        w = cov_inv.sum() /  cov_inv.sum().sum()
        return w


min_vol, weights, trades = run_backtest('month', '2006-1-1')

min_vol = min_vol.rename('MinVol')

t = pd.DataFrame(min_vol).join(RET[['SPY','TLT','GLD']])
```

```
t[100:].add(1).cumprod().plot()
t[100:].add(1).cumprod().plot(logy=True)
```

Out[4]: <AxesSubplot:>





Compare statistics for this table:

In [5]:

```
annual_returns = t[:'2020'].add(1).resample('A').prod().sub(1)
r_annual_Tbill = RATES.Treasury_1.resample('A').first()

x = pd.DataFrame()
x['Average_returns']   = annual_returns.mean()
x['Geometric_average'] = annual_returns.add(1).prod().pow(1/len(annual_returns))
x['Risk_premium']      = annual_returns.sub(r_annual_Tbill, 'rows').dropna().mea
x['Volatility']        = t[:'2020'].std() * 252**0.5
x['Sharpe_ratio']      = x.Risk_premium / x.Volatility
x
```

Out[5]:

	Average_returns	Geometric_average	Risk_premium	Volatility	Sharpe_ratio
MinVol	0.100066	0.097463	0.085513	0.082999	1.030285
SPY	0.112827	0.097954	0.098274	0.200886	0.489203
TLT	0.081284	0.070236	0.066731	0.144719	0.461107
GLD	0.098515	0.086228	0.083962	0.185648	0.452264

Average return review:

In [6]:

```
r1 =  0.1
r2 = -0.1

value_1_dollar_invested = (1+r1)*(1+r2)
compound_ret            = (1+r1)*(1+r2) - 1
average_ret             = (r1 + r2) / 2
geometric_average_ret   = ((1+r1)*(1+r2))**(1/2) - 1

print('value_1_dollar_invested:', round(value_1_dollar_invested, 5))
print('compound_ret:',            round(compound_ret,            5))
print('average_ret:',             round(average_ret,             5))
print('geometric_average_ret:',   round(geometric_average_ret,   5))
```

```
value_1_dollar_invested: 0.99
compound_ret: -0.01
average_ret: 0.0
geometric_average_ret: -0.00501
```

Interpretation of returns:

- geometric average represents performance in sample (if we compound this average we get the total compound return)
- the difference between the arithmetic average and the geometric average increases if the volatility increases
- the arithmetic average is the best estimator of the expected return
- future compound returns have more possible upside than downside (at most you lose 100%, but you can gain a lot more and 100%)
- because future compound returns are skewed towards the upside, the expected future return is higher than the median future return
- We can use the sample arithmetic average to estimate the expected future return and the geometric average to estimate the median future return

Let's put the calculation of the return statistics and the plots into a function:

In [7]:

```
def compare_performance(t):
    t.add(1).cumprod().plot()
    t.add(1).cumprod().plot(logy=True)

    annual_returns = t[:'2020'].add(1).resample('A').prod().sub(1)
    r_annual_Tbill = RATES.Treasury_1.resample('A').first()

    x = pd.DataFrame()
    x['Average_returns']   = annual_returns.mean()
    x['Geometric_average'] = annual_returns.add(1).prod().pow(1/len(annual_retur
    x['Risk_premium']      = annual_returns.sub(r_annual_Tbill, 'rows').dropna()
    x['Volatility']        = t[:'2020'].std() * 252**0.5
    x['Sharpe_ratio']      = x.Risk_premium / x.Volatility

    return x
```

Backtest with leverage

Example weights with leverage:

In [8]:
```
w = pd.Series({'SPY':0.6,'TLT':0.4})

w.multiply(1.5)
```

Out[8]:
```
SPY     0.9
TLT     0.6
dtype: float64
```

Margin loan:

In [10]:
```
pd.Series({'MarginLoan':-0.5})
```

Out[10]:
```
MarginLoan   -0.5
dtype: float64
```

Add the margin loan to weights:

In [11]:
```
w.multiply(1.5).append(pd.Series({'MarginLoan':-0.5}))
```

Out[11]:
```
SPY           0.9
TLT           0.6
MarginLoan   -0.5
dtype: float64
```

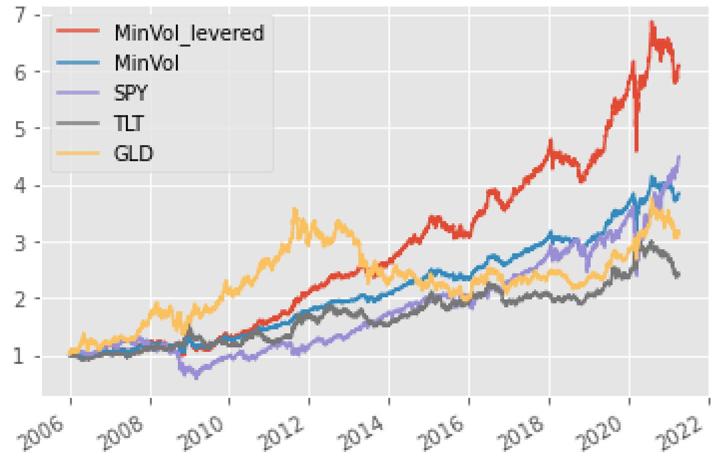
(Note: weights must sum to one!)

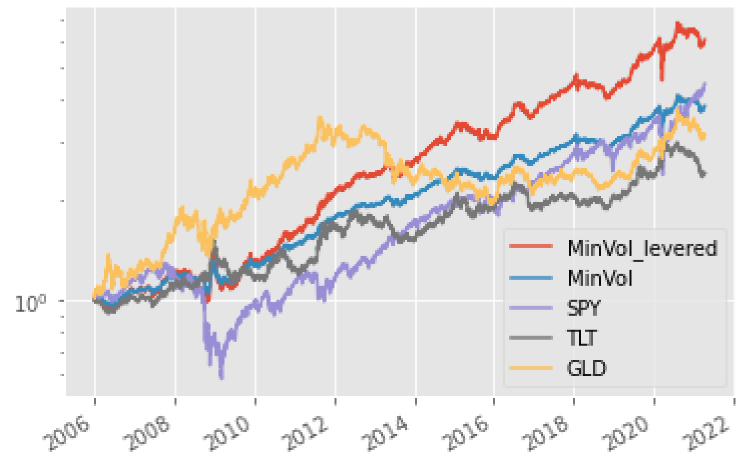
Put the leverage into the "select_weights" function:

In [12]:
```
def select_weights(date):
    cov     = RET[['SPY','TLT','GLD']][:date][-100:].cov() * 252
    cov_inv = pd.DataFrame(np.linalg.inv(cov), columns=cov.columns, index=cov.in

    w = cov_inv.sum() /  cov_inv.sum().sum()                         # Min-vol weigh

    w = w.multiply(1.5).append(pd.Series({'MarginLoan':-0.5}))   # Weights with
    return w


min_vol_levered, weights, trades = run_backtest('month','2006-1-1')

min_vol_levered = min_vol_levered.rename('MinVol_levered')

t = pd.DataFrame(min_vol_levered).join(min_vol).join(RET[['SPY','TLT','GLD']])

compare_performance(t)
```

Out[12]:

	Average_returns	Geometric_average	Risk_premium	Volatility	Sharpe_ratio
MinVol_levered	0.139858	0.133697	0.125304	0.124852	1.003620
MinVol	0.100066	0.097463	0.085513	0.082999	1.030285
SPY	0.112827	0.097954	0.098274	0.200886	0.489203
TLT	0.081284	0.070236	0.066731	0.144719	0.461107
GLD	0.098515	0.086228	0.083962	0.185648	0.452264





Portfolio weights:

In [13]:
```
weights[-3:]
```

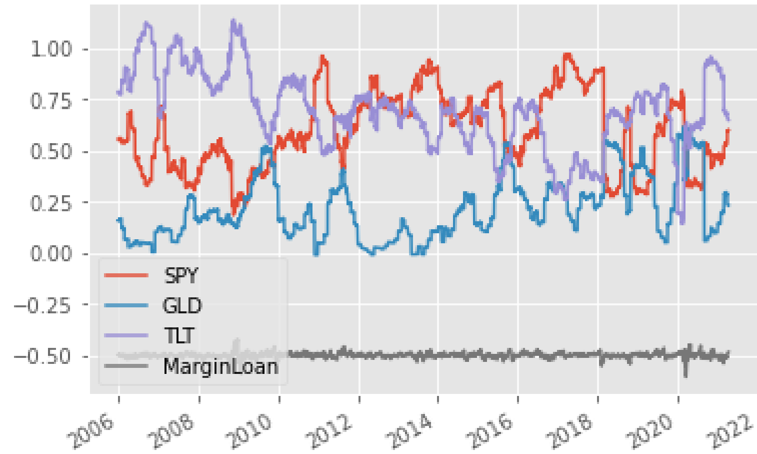
Out[13]:

	SPY	GLD	TLT	MarginLoan
2021-04-08	0.595699	0.235237	0.650613	-0.481549
2021-04-09	0.599922	0.233422	0.648139	-0.481483
2021-04-12	0.601042	0.232355	0.648829	-0.482227

Plot the weights:

In [14]:
```
weights.plot()
```

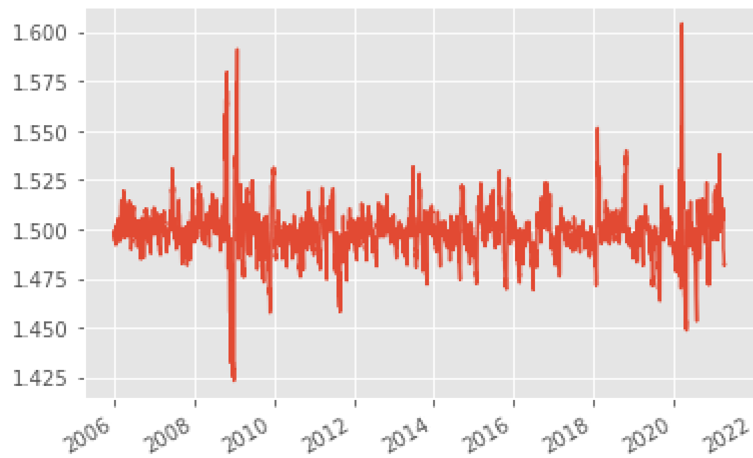
Out[14]: <AxesSubplot:>



Calculate leverage ratio:

In [15]:
```
weights.MarginLoan.abs().add(1).plot()
```

Out[15]: <AxesSubplot:>

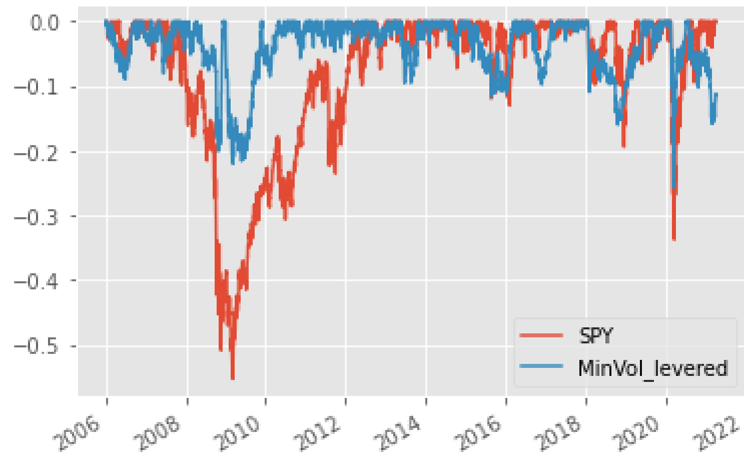


(For example, if MarginLoan = -0.8, we borrow 80% and put 180% into assets, so leverage = absolute value of MarginLoan plus one.)

Compare drawdowns:

In [16]:
```
hwm = t.add(1).cumprod().cummax()                    # high water mark
drawdown = t.add(1).cumprod()/hwm - 1.0              # % portfolio loss relative to
drawdown[['SPY','MinVol_levered']].plot()
```

Out[16]: <AxesSubplot:>


0.0
−0.1
−0.2
−0.3
−0.4
−0.5
SPY
MinVol_levered
2006
2008
2010
2012
2014
2016
2018
2020
2022