# Leverage I: Buying the Market on Margin

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 historical market data from Kenneth French's website:

In [2]:

```
url = 'http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/ftp/F-F_Research_D
r = requests.get(url)                                                            # Get file
z = zipfile.ZipFile(io.BytesIO( r.content ))                                     # Unzip fi
crsp = pd.read_csv( z.open(z.namelist()[0]), skiprows=4, index_col=0)[:-1] / 100
crsp.index = pd.to_datetime(crsp.index)                                          # Interpre
crsp
```

Out[2]:

| | Mkt-RF | SMB | HML | RF |
|---|---|---|---|---|
| **1926-07-01** | 0.0010 | -0.0024 | -0.0028 | 0.00009 |
| **1926-07-02** | 0.0045 | -0.0032 | -0.0008 | 0.00009 |
| **1926-07-06** | 0.0017 | 0.0027 | -0.0035 | 0.00009 |
| **1926-07-07** | 0.0009 | -0.0059 | 0.0003 | 0.00009 |
| **1926-07-08** | 0.0021 | -0.0036 | 0.0015 | 0.00009 |
| **...** | ... | ... | ... | ... |
| **2021-02-22** | -0.0112 | -0.0009 | 0.0314 | 0.00000 |
| **2021-02-23** | -0.0015 | -0.0128 | 0.0090 | 0.00000 |
| **2021-02-24** | 0.0115 | 0.0120 | 0.0134 | 0.00000 |
| **2021-02-25** | -0.0273 | -0.0112 | 0.0087 | 0.00000 |
| **2021-02-26** | -0.0028 | 0.0072 | -0.0156 | 0.00000 |

24934 rows × 4 columns

Add risk-free rate to excess returns:

In [3]:
```
crsp = (crsp['Mkt-RF'] + crsp['RF']).to_frame('CRSP')
crsp
```

Out[3]:

| | CRSP |
|---|---|
| **1926-07-01** | 0.00109 |
| **1926-07-02** | 0.00459 |
| **1926-07-06** | 0.00179 |
| **1926-07-07** | 0.00099 |
| **1926-07-08** | 0.00219 |
| **...** | ... |
| **2021-02-22** | -0.01120 |
| **2021-02-23** | -0.00150 |
| **2021-02-24** | 0.01150 |
| **2021-02-25** | -0.02730 |
| **2021-02-26** | -0.00280 |

24934 rows × 1 columns

GET SPY and VTI (Vanguard Total Stock Market Index Fund) for comparison:

In [4]:
```
PRICE       = tiingo.get_dataframe(['SPY', 'VTI'],'1900-01-01', metric_name='adj
PRICE.index = pd.to_datetime(PRICE.index).tz_convert(None)

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

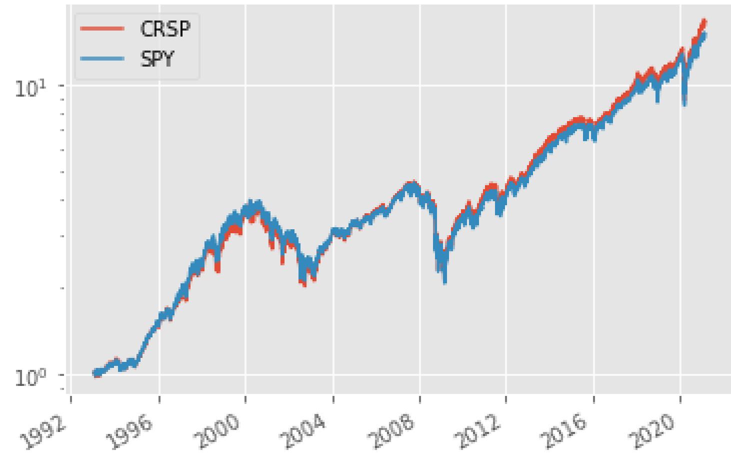
Out[4]:

| | SPY | VTI |
|---|---|---|
| **2021-03-25** | 0.005626 | 0.007372 |
| **2021-03-26** | 0.016115 | 0.016343 |
| **2021-03-29** | -0.000505 | -0.004698 |

CRSP index vs SPY:

In [5]:
```
crsp.join(RET.SPY).dropna().add(1).cumprod().plot(logy=True)
```

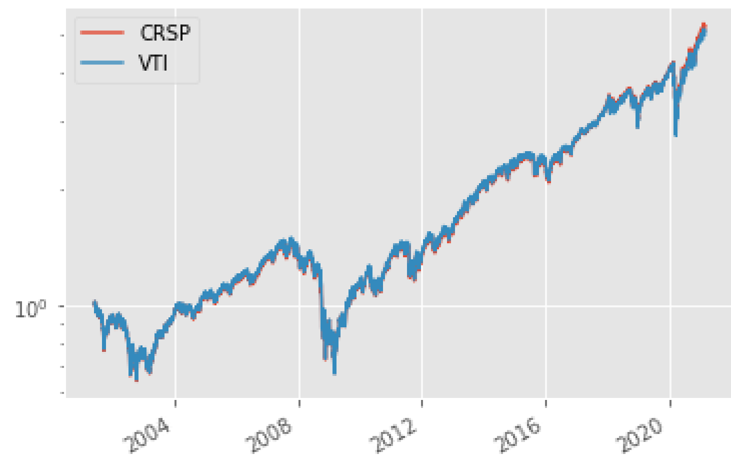
Out[5]: <AxesSubplot:>




CRSP index vs VTI:

In [6]:
```
crsp.join(RET.VTI).dropna().add(1).cumprod().plot(logy=True)
```

Out[6]: <AxesSubplot:>




Note: "Vanguard Total Stock Market ETF seeks to track the performance of the CRSP US Total Market Index" from here.

Interactive Brokers margin rates
Get federal funds rate:

In [7]:
```
fedfunds = quandl.get(['FRED/FEDFUNDS']).rename(columns={'FRED/FEDFUNDS - Value'
fedfunds
```

Out[7]:

| | Fedfunds |
|---|---|
| **Date** | |
| **1954-07-01** | 0.000032 |
| **1954-08-01** | 0.000048 |
| **1954-09-01** | 0.000042 |
| **1954-10-01** | 0.000034 |
| **1954-11-01** | 0.000033 |

| | Fedfunds |
|---|---|
| **Date** | |
| **...** | ... |
| **2020-10-01** | 0.000004 |
| **2020-11-01** | 0.000004 |
| **2020-12-01** | 0.000004 |
| **2021-01-01** | 0.000004 |
| **2021-02-01** | 0.000003 |

800 rows × 1 columns

Merge CRSP and federal funds rate:
(we use an 'outer' join since crsp table contains only trading dates and the dates in the fedfunds table are always the 1st of the month which might not be a trading day.)

In [9]:

```
data = crsp.join(fedfunds, how='outer')        # data has all dates from both tab
data['Fedfunds'] = data.Fedfunds.ffill()       # Forward fill fed funds rate to r
data = data.dropna()                           # drop all rows where we have miss
data
```

Out[9]:

| | CRSP | Fedfunds |
|---|---|---|
| **1954-07-01** | 0.00022 | 0.000032 |
| **1954-07-02** | 0.00992 | 0.000032 |
| **1954-07-06** | 0.00862 | 0.000032 |
| **1954-07-07** | 0.00022 | 0.000032 |
| **1954-07-08** | -0.00148 | 0.000032 |
| **...** | ... | ... |
| **2021-02-22** | -0.01120 | 0.000003 |
| **2021-02-23** | -0.00150 | 0.000003 |
| **2021-02-24** | 0.01150 | 0.000003 |
| **2021-02-25** | -0.02730 | 0.000003 |
| **2021-02-26** | -0.00280 | 0.000003 |

16780 rows × 2 columns

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

In [10]:

```
data['MarginRate'] = data.Fedfunds +  0.01/252  # divide by 252 to get daily spr
data[-3:]
```

Out[10]:

| | CRSP | Fedfunds | MarginRate |
|---|---|---|---|
| **2021-02-24** | 0.0115 | 0.000003 | 0.000043 |

| | CRSP | Fedfunds | MarginRate |
|---|---|---|---|
| **2021-02-25** | -0.0273 | 0.000003 | 0.000043 |
| **2021-02-26** | -0.0028 | 0.000003 | 0.000043 |

Returns with leverage example:

account value: 100
in stocks: 120
cash: -20 (loan)

Dollar tomorrow:

$$120 \times (1 + r_{\text{stocks}}) - 20 \times (1 + r_{\text{margin}})$$

Portfolio return:

$$\frac{120}{100} \times r_{\text{stocks}} - \frac{20}{100} \times r_{\text{margin}}$$

150% leverage with daily rebalancing:

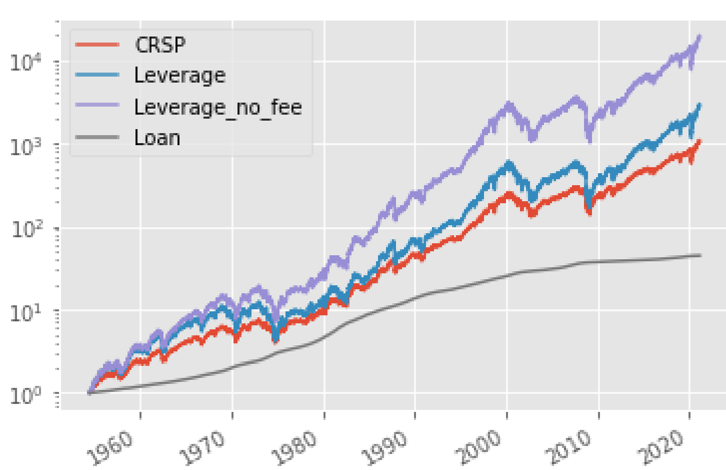
In [11]:
```
w = 1.5          # leverage ratio

t = pd.DataFrame()
t['CRSP']             = data.CRSP
t['Leverage']         = w * data.CRSP + (1-w) * data.MarginRate
t['Leverage_no_fee'] = w * data.CRSP + (1-w) * 0
t['Loan']             = data.MarginRate

t.add(1).cumprod()[-1:]    # Show most recent values
```

Out[11]:

| | CRSP | Leverage | Leverage_no_fee | Loan |
|---|---|---|---|---|
| **2021-02-26** | 1052.382754 | 2784.854016 | 18566.249141 | 44.470921 |

In [12]:
```
t.add(1).cumprod().plot(logy=True)
```

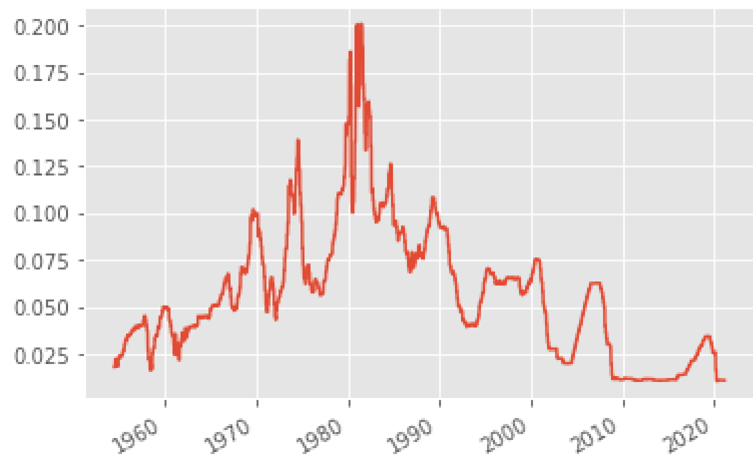
Out[12]: <AxesSubplot:>



Historical margin rate:

In [13]:
```
data.MarginRate.multiply(252).plot()
```

Out[13]: <AxesSubplot:>

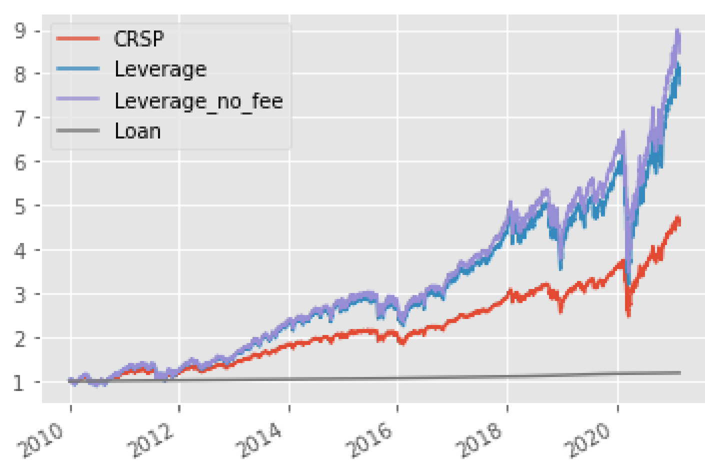



→ very high margin rates result in low return for levered strategy.

Leverage performance since 2010:

In [14]:
```
t['2010':].add(1).cumprod().plot()
```

Out[14]: <AxesSubplot:>

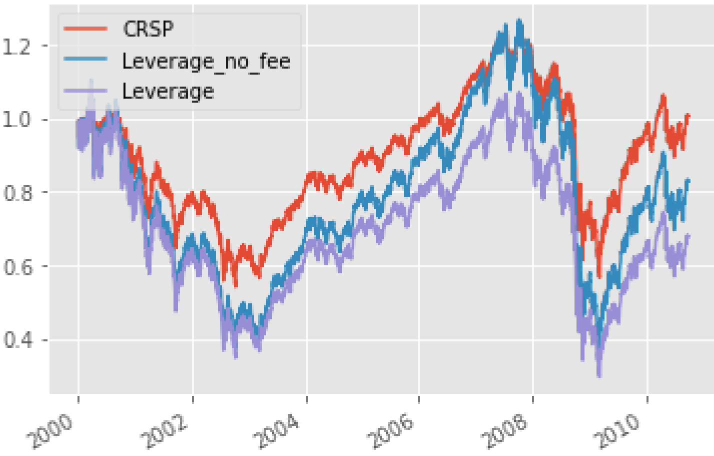



→ Best posibble combination: market up and margin rates low!

From January 2000 until September 2010:

In [15]:
```
t['2000':'2010-9'][['CRSP','Leverage_no_fee', 'Leverage']].add(1).cumprod().plot
```

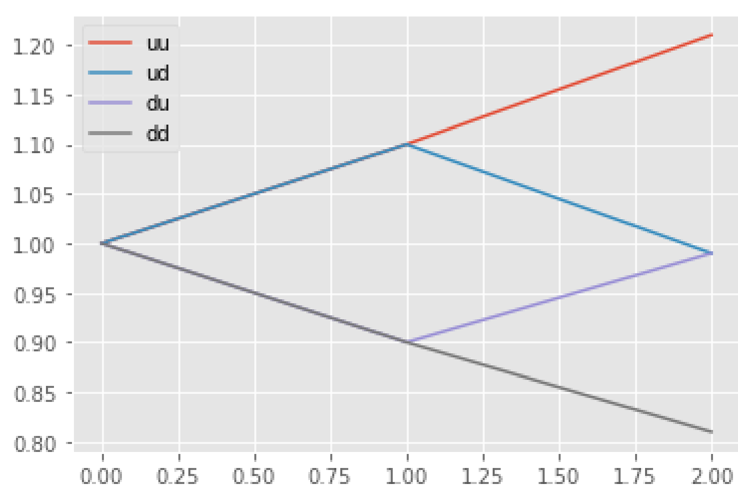
Out[15]: <AxesSubplot:>




Why does the levered portfolio underperform the market in this sample (even before fees)?
Binomial example:

In [16]:
```
# Suppose a stock goes up by 10% or down by 10%:
u =  0.1                  # Up factor
d = -0.1                  # Down factor

t = pd.DataFrame()
t['uu'] = [0,u,u]        # Twice up
t['ud'] = [0,u,d]        # First up, then down
t['du'] = [0,d,u]        # First down, then up
t['dd'] = [0,d,d]        # Twice down

t.add(1).cumprod().plot()
```

Out[16]: <AxesSubplot:>




In [17]:
```
t.add(1).cumprod()
```

Out[17]:

| | uu | ud | du | dd |
|---|---|---|---|---|
| **0** | 1.00 | 1.00 | 1.00 | 1.00 |
| **1** | 1.10 | 1.10 | 0.90 | 0.90 |
| **2** | 1.21 | 0.99 | 0.99 | 0.81 |

→ in the middle scenario (ud or du) the market does not return to starting value (because, for example if the stock goes down 10%, we need to go uo by more than 10% to get back to the starting value).

Suppose we use maximum leverage and we pay zero margin rate (so we get +/- 20% return instead of +/- 10%):

In [18]:

```
t.multiply(2).add(1).cumprod()
```

Out[18]:

| | uu | ud | du | dd |
|---|---|---|---|---|
| **0** | 1.00 | 1.00 | 1.00 | 1.00 |
| **1** | 1.20 | 1.20 | 0.80 | 0.80 |
| **2** | 1.44 | 0.96 | 0.96 | 0.64 |

→ note how we underperform the market in middle scenario (ud or du).

If we want the final value of the up-down and down-up paths to be equal to the starting value ($1), the stock needs to go up by 11.11%:

In [19]:

```
u = 0.111111            # Up factor
d = -0.1                # Down factor

t = pd.DataFrame()
t['uu'] = [0,u,u]
t['ud'] = [0,u,d]
t['du'] = [0,d,u]
t['dd'] = [0,d,d]

t.add(1).cumprod()
```

Out[19]:

| | uu | ud | du | dd |
|---|---|---|---|---|
| **0** | 1.000000 | 1.000000 | 1.0 | 1.00 |
| **1** | 1.111111 | 1.111111 | 0.9 | 0.90 |
| **2** | 1.234568 | 1.000000 | 1.0 | 0.81 |

With same leverage as above:

In [20]:

```
t.multiply(2).add(1).cumprod()
```

Out[20]:

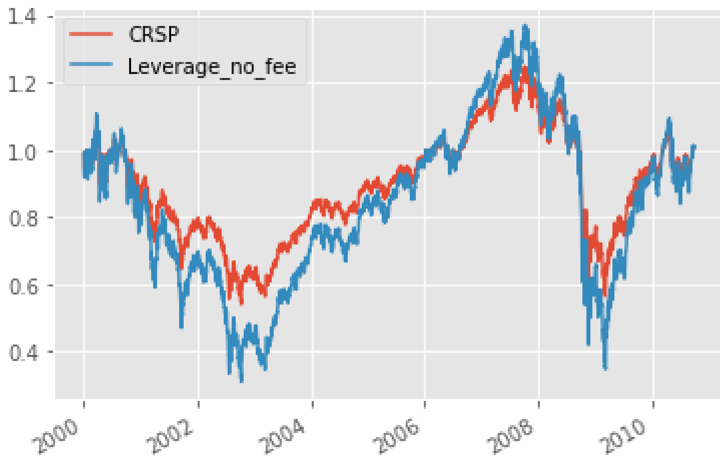
| | uu | ud | du | dd |
|---|---|---|---|---|
| **0** | 1.000000 | 1.000000 | 1.000000 | 1.00 |
| **1** | 1.222222 | 1.222222 | 0.800000 | 0.80 |
| **2** | 1.493827 | 0.977778 | 0.977778 | 0.64 |

→ leverage still underperforms in the middle scenarios!

Same time period, never rebalance:

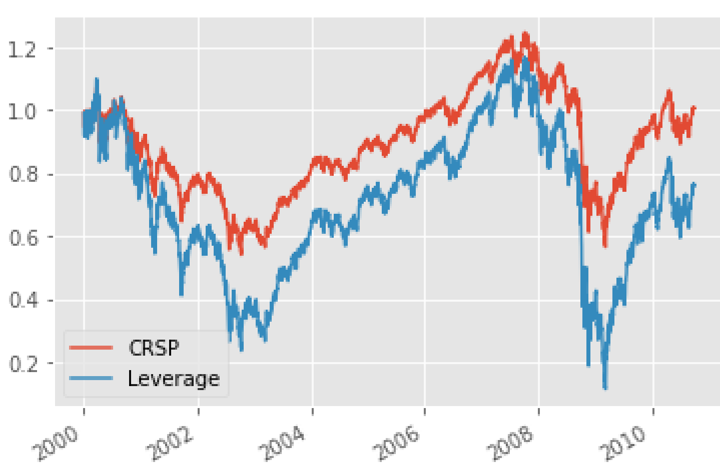
In [21]:
```python
w = 1.5

t = pd.DataFrame()
t['CRSP']             = data['2000':'2010-9'].CRSP.add(1).cumprod()
t['Leverage_no_fee'] = w * data['2000':'2010-9'].CRSP.add(1).cumprod() + (1-w) *

t.plot()
```

Out[21]: <AxesSubplot:>




In [22]:
```python
w = 1.5

t = pd.DataFrame()
t['CRSP']     = data['2000':'2010-9'].CRSP.add(1).cumprod()
t['Leverage'] = w * data['2000':'2010-9'].CRSP.add(1).cumprod() + (1-w) * data['

t.plot()
```

Out[22]: <AxesSubplot:>




In [ ]: