Selecting Assets IV: Filtering by Beta and Revenue

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

import os

# 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
```

Function for SEC data:

In [2]:

```
def ffill_values(item, dates):
    data = item.unstack('cik')
    data = data.reindex(dates.union(data.index)).sort_index()           # Add sp
    filing_dates = pd.read_csv('data/sec/dates/filing_dates.csv', index_col='cik
    last_filing_date_all_firms = filing_dates.max()                     # Most r

    for cik in data.columns:                                            # Loop o
        last_filing_date      = pd.Series(filing_dates[cik]).iloc[-1]   # Last d
        days_since_last_filed = (last_filing_date_all_firms - last_filing_date).
        last_date_this_firm   = dates[-1] if days_since_last_filed < 120 else la
        data.loc[:last_date_this_firm, cik].ffill(inplace=True)         # Forwar

    return data.loc[dates]                                              # Return
```

Our rebalance function:

In [3]:

```
def get_rebalance_dates(frequency, start_date):
    price = PRICE[PRICE.index>start_date]
    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'):

    rebalance_dates = get_rebalance_dates(frequency, backtest_start)

    portfolio_value = pd.Series(1,                             index=[rebalance_dates
```

```
    weights          = pd.DataFrame(columns=PRICE.columns, index=[rebalance_dates
    trades           = pd.DataFrame(columns=PRICE.columns, index=[rebalance_dates

    previous_positions = weights.iloc[0]

    for i in range(1, 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()

        assets        = select_assets(start_date)                # Call "select_a
        start_weights = select_weights(start_date, assets)       # Call "select_w

        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
```

Get data

Get sales data:

In [4]:
```
sales = pd.read_csv('data/sec/items/Sales.csv',  parse_dates=['filed'], index_co
```

Now forward-fill the sales data to all trading days and rename the columns to ticker symbols:

In [5]:
```
trading_days = pd.to_datetime( tiingo.get_dataframe('SPY','2009-04-15').index ).

salesQ = ffill_values( sales.valueQ,  trading_days )
salesA = ffill_values( sales.valueA,  trading_days )


symbols = pd.read_csv('data/ticker_symbols/symbols.csv',index_col=0)

SALESQ = salesQ.rename(columns=symbols.ticker)
SALESA = salesA.rename(columns=symbols.ticker)
```

Get price data:

In [6]:
```
PRICE  = pd.read_csv('data/tiingo/close.csv',    index_col='date', parse_dates=[
RET    = pd.read_csv('data/tiingo/adjClose.csv', index_col='date', parse_dates=[
VOLUME = pd.read_csv('data/tiingo/volume.csv',   index_col='date', parse_dates=[
DOLLAR_VOLUME = VOLUME * PRICE
```

Get benchmark:

In [7]:
```
vti = tiingo.get_dataframe(['VTI'], '1990-1-1', metric_name='adjClose')
vti.index = pd.to_datetime(vti.index).tz_convert(None)
vti_ret = vti.pct_change().VTI
vti_ret
```

Out[7]:
```
2001-05-31         NaN
2001-06-01    0.006969
2001-06-04    0.004325
2001-06-05    0.014643
2001-06-06   -0.008489
                ...
2021-04-23    0.012115
2021-04-26    0.003637
2021-04-27   -0.000459
2021-04-28   -0.000138
2021-04-29    0.003810
Name: VTI, Length: 5010, dtype: float64
```

Calculate betas

Calculate Apple beta:

In [8]:
```
RET.AAPL.cov(vti_ret)  # Apple covariance with VTI
```

Out[8]:
```
0.0001296223480592608
```

In [9]:
```
vti_ret[ RET.index[0]:RET.index[-1] ].var()     # VTI variance
```

Out[9]:
```
0.00012493567269040163
```

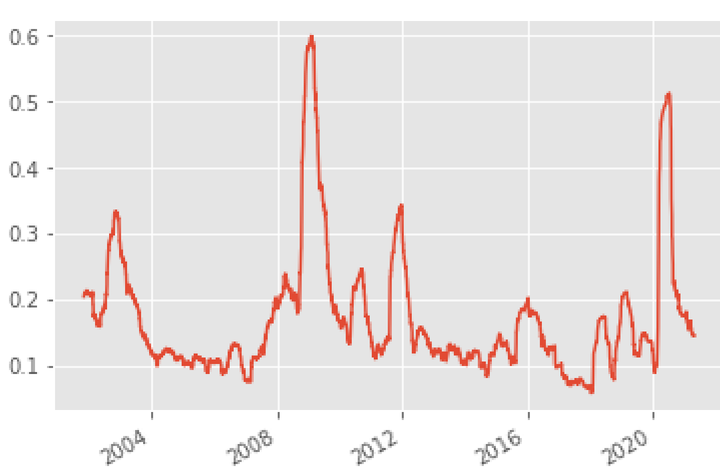
In [10]:
```
beta = RET.AAPL.cov(vti_ret) / vti_ret[ RET.index[0]:RET.index[-1] ].var()
beta
```

Out[10]:
```
1.0375127076833615
```

VTI rolling volatility:

In [11]:
```
vti_ret.rolling(100).std().multiply(252**0.5).plot()
```

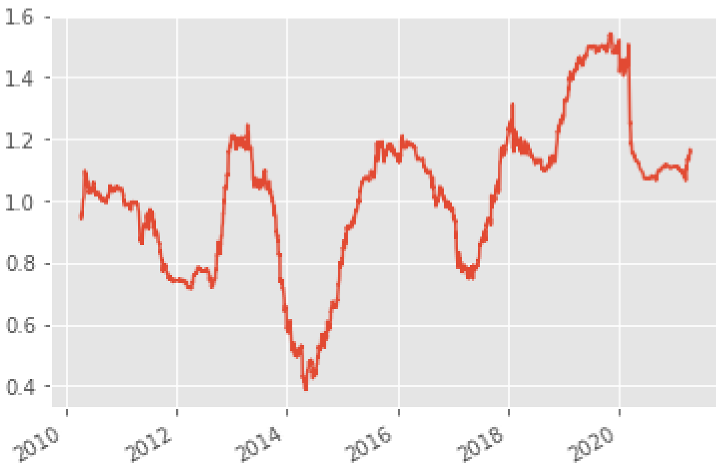
Out[11]:
```
<AxesSubplot:>
```



Apple rolling beta:

In [12]:
```
n = 252

cov = RET.AAPL.rolling(n).cov(vti_ret)
var = vti_ret.rolling(n).var()

(cov/var).plot()
```

Out[12]: <AxesSubplot:>



Calculate betas for all firms:

In [13]:
```
n = 252

firms = PRICE.columns
start = PRICE.index[0]
end   = PRICE.index[-1]

var = vti_ret.rolling(n).var()

BETA = pd.DataFrame()

for firm in firms:
    cov        = RET[firm].rolling(n).cov(vti_ret)
    BETA[firm] = cov / var

BETA = BETA[start:end]
BETA
```

Out[13]:

	AIR	ABT	WDDD	ACU	AE	BKTI	AMD	APD	CE(
2009-04-15	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
2009-04-16	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
2009-04-17	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
2009-04-20	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

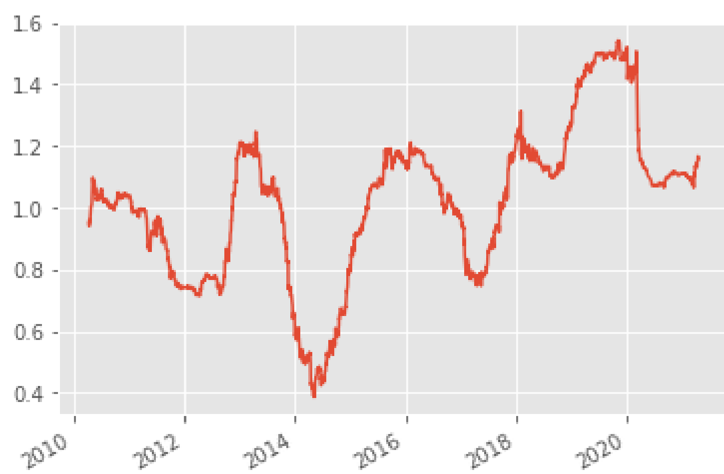
	AIR	ABT	WDDD	ACU	AE	BKTI	AMD	APD	CE(
2009-04-21	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
...	...	...	...	...	...	...	...	...	
2021-04-08	2.123954	0.653472	0.533141	0.446818	0.890821	0.465828	1.151228	0.905890	1.3628
2021-04-09	2.157330	0.642260	0.654154	0.416648	0.866064	0.388036	1.162636	0.874186	1.3703
2021-04-12	2.141830	0.642040	0.632524	0.423573	0.850962	0.320253	1.178321	0.874190	1.3490
2021-04-13	2.118640	0.644229	0.637885	0.426704	0.839602	0.462180	1.203143	0.873042	1.3446
2021-04-14	2.157536	0.627605	0.476072	0.426066	0.838726	0.626483	1.172730	0.877449	1.3634

3021 rows × 5914 columns

Example:

In [14]:
```
BETA.AAPL.plot()  # same graph as above
```

Out[14]: <AxesSubplot:>

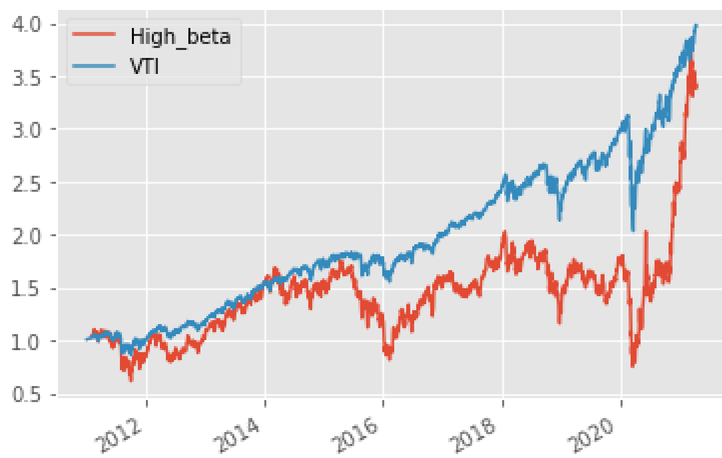


Backtest high-beta strategy:

In [15]:
```
def select_assets(date):
    all_firms = PRICE.columns

    p = PRICE        [all_firms].loc[date]
    v = DOLLAR_VOLUME[all_firms].loc[date]

    min_price  = p[p>1].index
    min_volume = v[v>100000].index

    tradable_assets = min_price.intersection(min_volume)

    assets = BETA[tradable_assets][:date].iloc[-1].nlargest(100).index     # 100
    return assets
```

```
def select_weights(date, assets):
    return pd.Series(1/len(assets), index=assets)



high_beta, weights, trades = run_backtest('month', '2011-1-1')

t = high_beta.to_frame('High_beta').join(vti_ret)
t.add(1).cumprod().plot()
```

Out[15]: <AxesSubplot:>



Calculate the beta of this portfolio:

In [16]:

```
high_beta.cov(vti_ret) / vti_ret[high_beta.index[0]:high_beta.index[-1]].var()
```

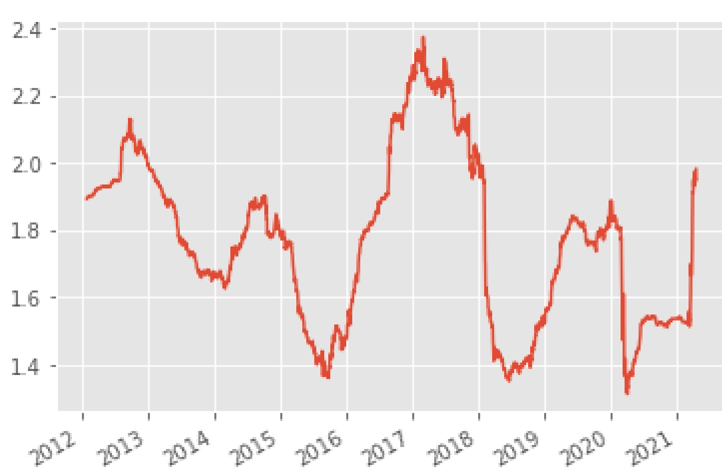
Out[16]: 1.707053675318617

Rolling beta of this portfolio:

In [17]:

```
cov = high_beta.rolling(252).cov(vti_ret)
var = vti_ret.rolling(252).var()
(cov / var).plot()
```

Out[17]: <AxesSubplot:>

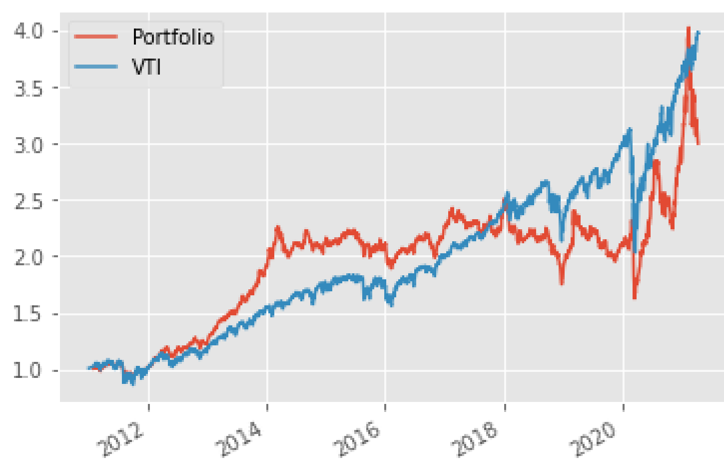


Low-beta strategy:

In [18]:

```
def select_assets(date):
    all_firms = PRICE.columns#.intersection(SALESA.columns)

    p = PRICE        [all_firms].loc[date]
    v = DOLLAR_VOLUME[all_firms].loc[date]

    min_price  = p[p>1].index
    min_volume = v[v>100000].index

    tradable_assets = min_price.intersection(min_volume)

    assets = BETA[tradable_assets][:date].iloc[-1].nsmallest(100).index
    return assets



low_beta, weights, trades = run_backtest('month', '2011-1-1')

t = low_beta.to_frame('Portfolio').join(vti_ret)
t.add(1).cumprod().plot()
```

Out[18]: <AxesSubplot:>



Calculate the beta of this portfolio:

In [19]:

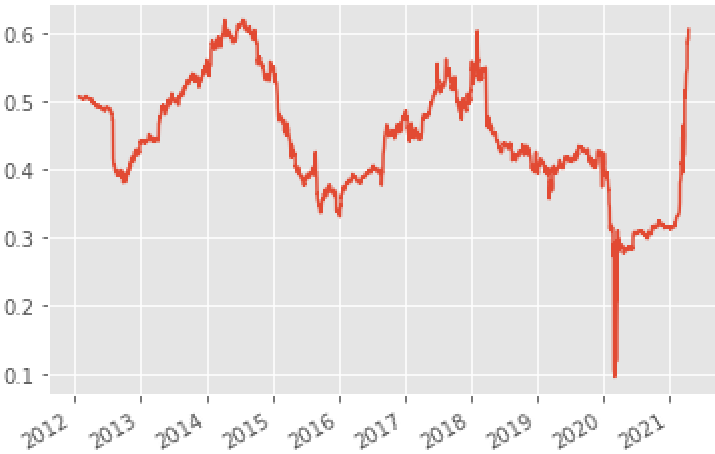
```
low_beta.cov(vti_ret) / vti_ret[low_beta.index[0]:low_beta.index[-1]].var()
```

Out[19]: 0.4300665269463516

In [20]:

```
cov = low_beta.rolling(252).cov(vti_ret)
var = vti_ret.rolling(252).var()
(cov / var).plot()
```

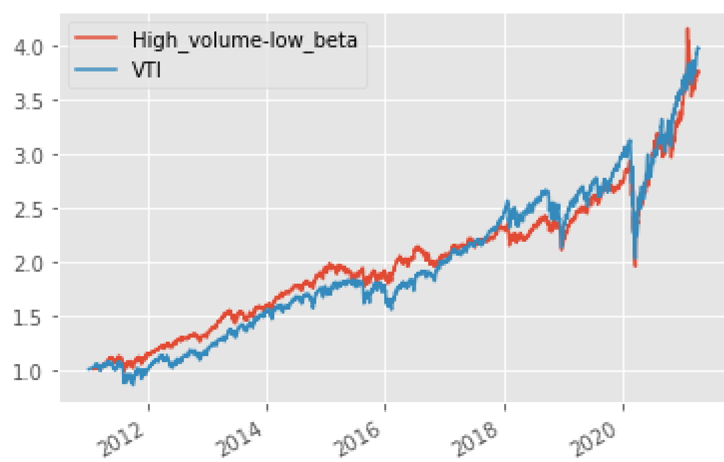
Out[20]: <AxesSubplot:>



Among 500 firms with highest average volume during the last 30 tradig days, select the 100 firms with lowest beta:

In [21]:
```
def select_assets(date):
    high_volume = DOLLAR_VOLUME[:date][-30:].mean().nlargest(500).index
    assets      = BETA[high_volume][:date].iloc[-1].nsmallest(100).index
    return assets



high_volume_low_beta, weights, trades = run_backtest('month', '2011-1-1')

t = high_volume_low_beta.to_frame('High_volume-low_beta').join(vti_ret)
t.add(1).cumprod().plot()
```

Out[21]:
```
<AxesSubplot:>
```



Volatilties:

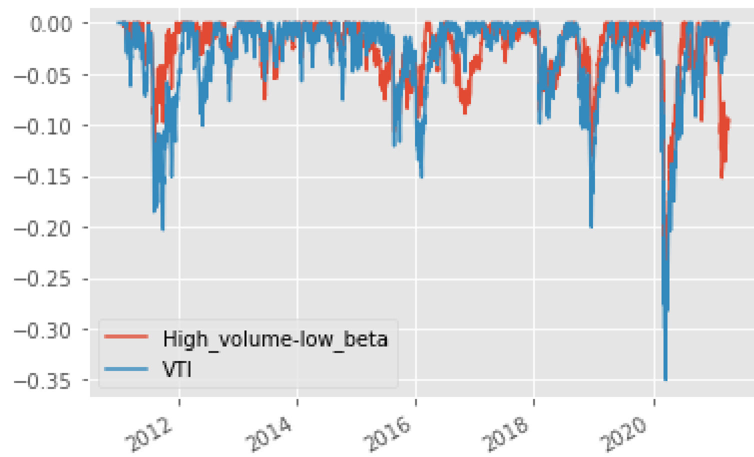
In [22]:
```
t.std()*252**0.5
```

Out[22]:
```
High_volume-low_beta    0.147351
VTI                     0.175747
dtype: float64
```

Compare drawdowns:

In [23]:
```
hwm = t.add(1).cumprod().cummax()          # cummax: maximum value from 1st row t

drawdown = t.add(1).cumprod()/hwm - 1.0    # % portfolio loss relative to most re
drawdown.plot()
```

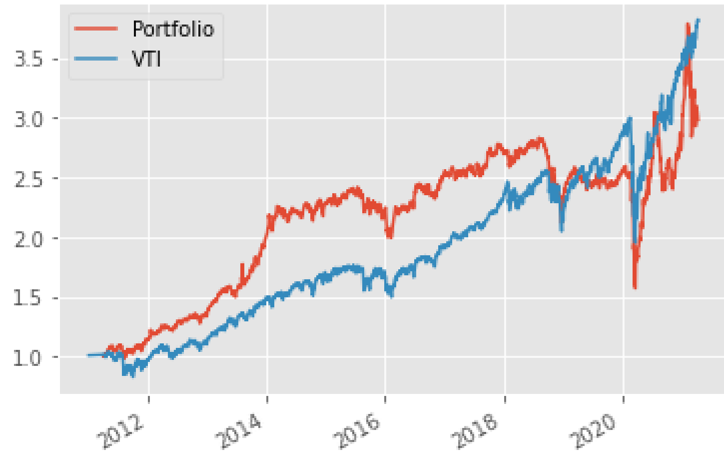
Out[23]: <AxesSubplot:>



Select 50 firms with highest annual sales among the 100 firms with lowest beta (among all tradable assets):

In [25]:
```
def select_assets(date):
    all_firms = PRICE.columns.intersection(SALESA.columns)  # All firms that in

    p = PRICE        [all_firms].loc[date]
    v = DOLLAR_VOLUME[all_firms].loc[date]

    min_price  = p[p>1].index
    min_volume = v[v>100000].index

    tradable_assets = min_price.intersection(min_volume)

    low_beta = BETA[tradable_assets][:date].iloc[-1].nsmallest(100).index
    assets   = SALESA[low_beta][:date].iloc[-1].nlargest(50).index
    return assets


portfolio, weights, trades = run_backtest('quarter', '2011-1-1')

t = portfolio.to_frame('Portfolio').join(vti_ret)
t.add(1).cumprod().plot()
```

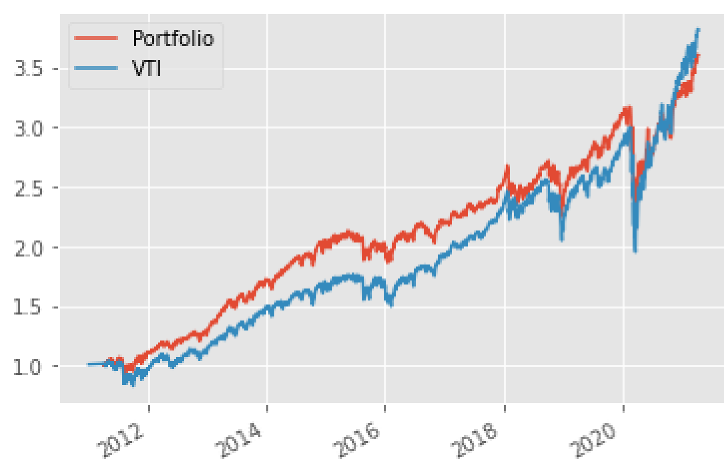
Out[25]: <AxesSubplot:>



Select 50 firms with lowest beta among the 100 firms with highest annual sales (among all tradable assets):

In [26]:
```
def select_assets(date):
    all_firms = PRICE.columns.intersection(SALESA.columns)

    p = PRICE        [all_firms].loc[date]
    v = DOLLAR_VOLUME[all_firms].loc[date]

    min_price  = p[p>1].index
    min_volume = v[v>100000].index

    tradable_assets = min_price.intersection(min_volume)

    high_sales = SALESA[tradable_assets][:date].iloc[-1].nlargest(100).index
    assets     = BETA  [high_sales]     [:date].iloc[-1].nsmallest(50).index
    return assets


portfolio, weights, trades = run_backtest('quarter', '2011-1-1')

t = portfolio.to_frame('Portfolio').join(vti_ret)
t.add(1).cumprod().plot()
```

Out[26]: <AxesSubplot:>

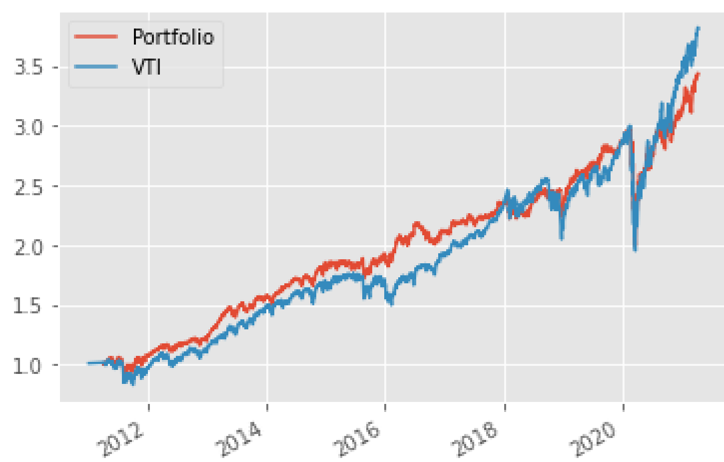


Select firms that have both low beta and high sales:

In [27]:

```
def select_assets(date):
    all_firms = PRICE.columns.intersection(SALESA.columns)

    p = PRICE        [all_firms].loc[date]
    v = DOLLAR_VOLUME[all_firms].loc[date]

    min_price  = p[p>1].index
    min_volume = v[v>100000].index

    tradable_assets = min_price.intersection(min_volume)

    #high_volume = DOLLAR_VOLUME[:date][-30:].mean().nlargest(500).index
    high_sales = SALESA[tradable_assets][:date].iloc[-1].nlargest(500).index
    low_beta   = BETA  [tradable_assets][:date].iloc[-1].nsmallest(500).index

    assets   = high_sales.intersection(low_beta)
    return assets


portfolio, weights, trades = run_backtest('quarter', '2011-1-1')

t = portfolio.to_frame('Portfolio').join(vti_ret)
t.add(1).cumprod().plot()
```

Out[27]: <AxesSubplot:>



How many assets?

In [28]:

```
weights
```

Out[28]:

	AIR	ABT	WDDD	ACU	AE	BKTI	AMD	APD	CECE	MATX	...	DMYI	AJAX	SPI
2011-01-03	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	N
2011-04-01	NaN	0.006617	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	N
2011-04-04	NaN	0.006722	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	N

	AIR	ABT	WDDD	ACU	AE	BKTI	AMD	APD	CECE	MATX	...	DMYI	AJAX	SPI
2011-04-05	NaN	0.006689	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	N
2011-04-06	NaN	0.006749	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	N
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2021-04-08	NaN	0.012640	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	N
2021-04-09	NaN	0.012759	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	N
2021-04-12	NaN	0.012738	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	N
2021-04-13	NaN	0.012918	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	N
2021-04-14	NaN	0.012834	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	N

2526 rows × 5914 columns

In [29]:

```
weights.count() # Count non-missing values vertically (across rows)
```

Out[29]:

```
AIR        0
ABT      701
WDDD       0
ACU        0
AE       126
        ...
NEBCU      0
PLTK       0
DDMXU      0
FPAC       0
GEG        0
Length: 5914, dtype: int64
```

In [30]:

```
weights.count('columns') # Count non-missing values horizontally (across columns
```

Out[30]:

```
2011-01-03      0
2011-04-01    151
2011-04-04    151
2011-04-05    151
2011-04-06    151
             ...
2021-04-08     78
2021-04-09     78
2021-04-12     78
2021-04-13     78
```

```
2021-04-14     78
Length: 2526, dtype: int64
```

In [31]:

```
weights[1:].count('columns').plot()
```

Out[31]: <AxesSubplot:>