Selecting Assets III: Tradable Securities

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
```

Get price data:

In [2]:
```
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 [3]:
```
vti = tiingo.get_dataframe(['VTI'], '1990-1-1', metric_name='adjClose')
vti.index = pd.to_datetime(vti.index).tz_convert(None)
```

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:

In [5]:
```
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



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 )
```

Now we need to change the column labels from CIKs to ticker symbols:

In [6]:
```
symbols = pd.read_csv('data/ticker_symbols/symbols.csv',index_col=0)

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

Backtest function:

In [7]:
```
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
```

Top ten highest annual sales, equal weight and rebalance quarterly:

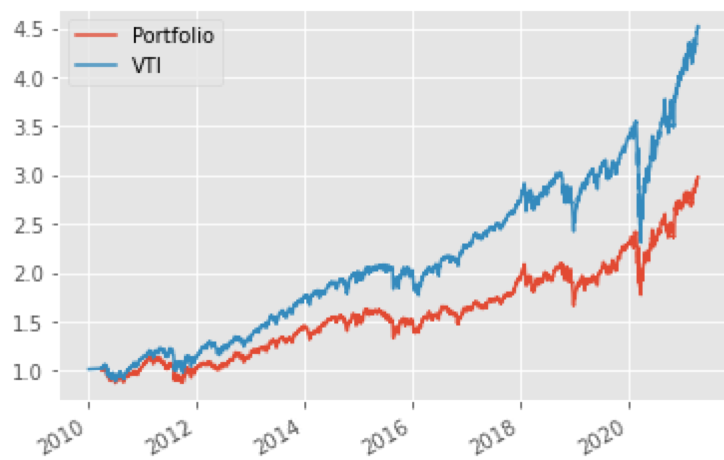
In [8]:
```
def select_assets(date):
    assets = SALESA[:date].iloc[-1].nlargest(10).index
    return assets


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


portfolio, weights, trades = run_backtest('quarter', '2010-1-1')

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

Out[8]: <AxesSubplot:>



Same strategy for top 100 firms:

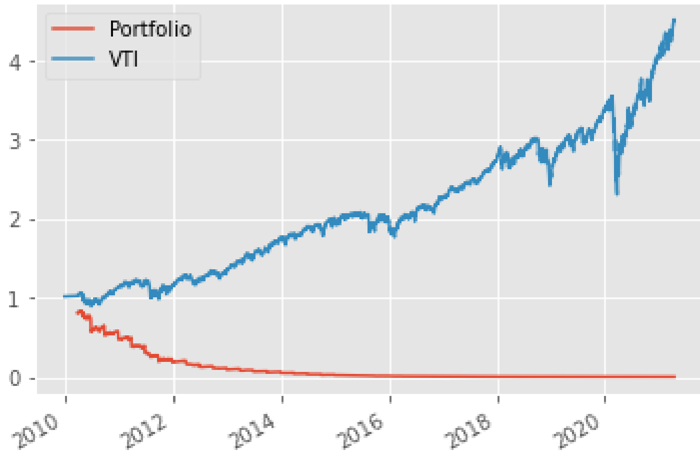
In [9]:
```
def select_assets(date):
    assets = SALESA[:date].iloc[-1].nlargest(100).index
    return assets



portfolio, weights, trades = run_backtest('quarter', '2010-1-1')

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

Out[9]: <AxesSubplot:>



Where does all the money go?
Problem: some firms from the SALES table (SEC data) by not be in the RET table (tiingo data). Any firm that we return from the "select_asset" function and that is not in the RET table will create a missing value and when we sum up the firm values to get the portfolio value, the missing firms will not be part of the sum and therefore count as zero.
Solution: find all firms that both tables have in common:

In [10]:
```
PRICE.columns
```

Out[10]:
```
Index(['AIR', 'ABT', 'WDDD', 'ACU', 'AE', 'BKTI', 'AMD', 'APD', 'CECE', 'MATX',
       ...
       'DMYI', 'AJAX', 'SPNV', 'IIAC', 'SPFR', 'NEBCU', 'PLTK', 'DDMXU',
       'FPAC', 'GEG'],
      dtype='object', length=5914)
```

In [11]:
```
SALESA.columns
```

Out[11]:
```
Index([  'AIR',    'ABT',   'WDDD',     2034,    'ACU',     'AE',   'BKTI',    'AMD',
          2491,    'APD',
       ...
        'CLNN',   'GBNY',   'SPFR',   'POSH',   'WETH', 1827855,   'PLTK',   'FORA',
         'GEG', 'GPACU'],
      dtype='object', name='cik', length=10144)
```

In [12]:
```
PRICE.columns.intersection(SALESA.columns)
```

Out[12]:
```
Index(['AIR', 'ABT', 'WDDD', 'ACU', 'AE', 'BKTI', 'AMD', 'APD', 'CECE', 'MATX',
       ...
       'APSG', 'AFRM', 'GHLD', 'AAN', 'LESL', 'FHTX', 'GBNY', 'SPFR', 'PLTK',
       'GEG'],
      dtype='object', length=5013)
```

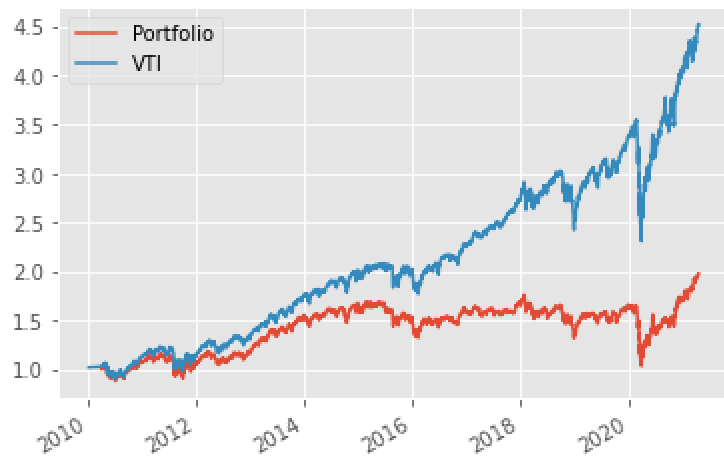
And now restrict our selection to these common firms:

In [13]:
```
def select_assets(date):
    all_firms = PRICE.columns.intersection(SALESA.columns)
    assets = SALESA[ all_firms ][:date].iloc[-1].nlargest(100).index
    return assets


portfolio, weights, trades = run_backtest('quarter', '2010-1-1')
```

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

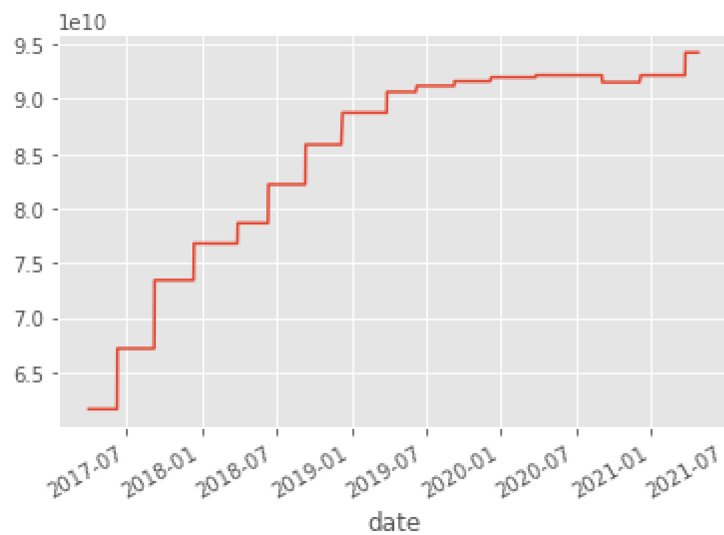
Out[13]: <AxesSubplot:>



Looks better, but the return still seems to low (the top 100 firms by revenue should perform similar to the market).
Problem: some of the firms might report Sales even though they currently are not publicly traded.
Example:

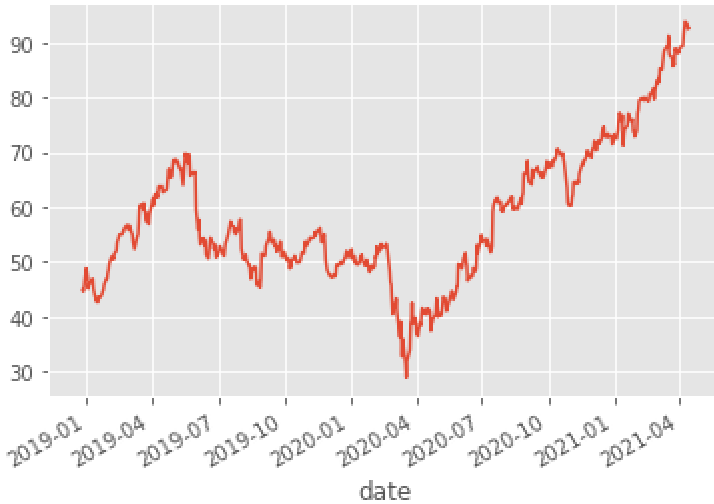
In [14]:
```
SALESA['DELL'].plot()
```

Out[14]: <AxesSubplot:xlabel='date'>



In [15]:
```
PRICE['DELL'].plot()
```

Out[15]: <AxesSubplot:xlabel='date'>



Find all firms that have a price on a specific date:

In [19]:
```
date = '2017-6-30'

all_firms = PRICE.columns.intersection(SALESA.columns)

PRICE[all_firms].loc[date]   # All firms on this day
```

Out[19]:
```
AIR      34.76
ABT      48.61
WDDD      0.03
ACU      28.60
AE       41.08
         ...
FHTX       NaN
GBNY       NaN
SPFR       NaN
PLTK       NaN
GEG        NaN
Name: 2017-06-30 00:00:00, Length: 5013, dtype: float64
```

In [20]:
```
PRICE[all_firms].loc[date].dropna()  # drop firms with missing prices
```

Out[20]:
```
AIR       34.76
ABT       48.61
WDDD       0.03
ACU       28.60
AE        41.08
          ...
ASRT      10.74
PRG       38.90
BNTC       1.85
TPL      293.78
FMHS       0.02
Name: 2017-06-30 00:00:00, Length: 4057, dtype: float64
```

In [21]:
```
PRICE[all_firms].loc[date].dropna().index
```

Out[21]:
```
Index(['AIR', 'ABT', 'WDDD', 'ACU', 'AE', 'BKTI', 'AMD', 'APD', 'CECE', 'MATX',
       ...
       'IAC', 'ANAT', 'XPER', 'BLAB', 'HIGR', 'ASRT', 'PRG', 'BNTC', 'TPL',
```

```
        'FMHS'],
      dtype='object', length=4057)
```

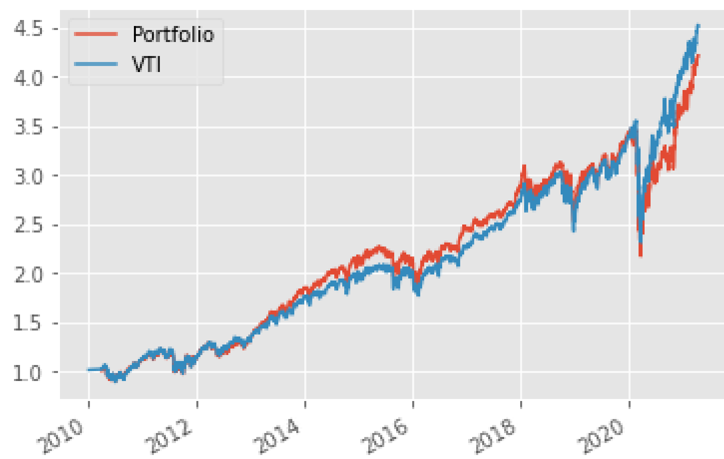
Now try top 100 firms by sales again:

In [22]:
```
def select_assets(date):
    all_firms          = PRICE.columns.intersection(SALESA.columns)
    assets_with_price = PRICE[all_firms].loc[date].dropna().index
    assets             = SALESA[ assets_with_price ][:date].iloc[-1].nlargest(100
    return assets


portfolio, weights, trades = run_backtest('quarter', '2010-1-1')

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

Out[22]: `<AxesSubplot:>`



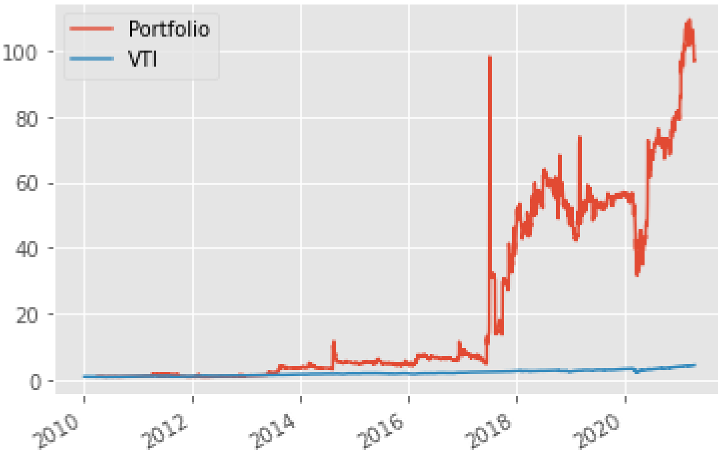
Small stocks:

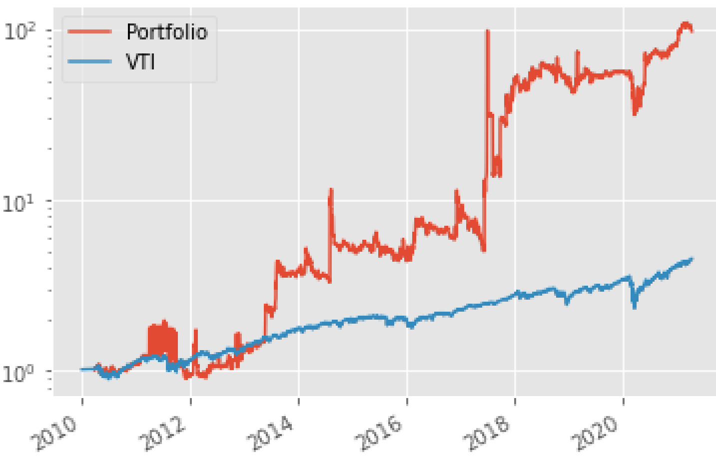
In [23]:
```
def select_assets(date):
    all_firms          = PRICE.columns.intersection(SALESA.columns)
    assets_with_price = PRICE[all_firms].loc[date].dropna().index
    assets = SALESA[ assets_with_price ][:date].iloc[-1].nsmallest(10).index
    return assets


portfolio, weights, trades = run_backtest('quarter', '2010-1-1')

t = portfolio.to_frame('Portfolio').join(vti.pct_change())
t.add(1).cumprod().plot()
t.add(1).cumprod().plot(logy=True)
```

Out[23]: `<AxesSubplot:>`





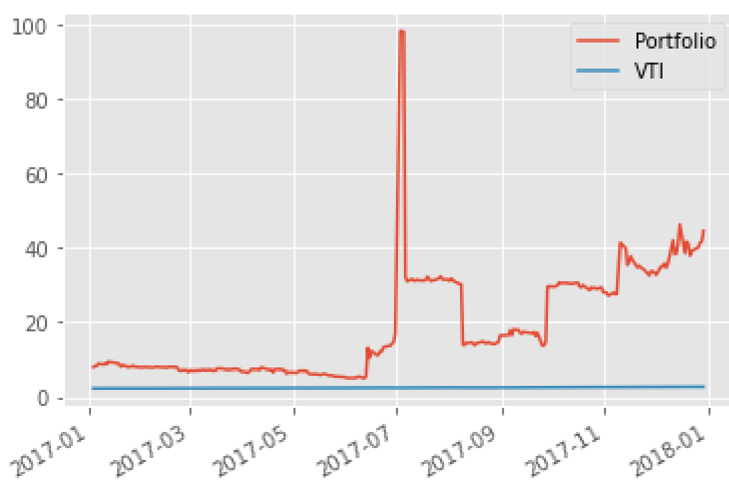
Too go to be true?

Check the spike where the value goes from ≈ 0 to 100 and then back to below 20:

In [24]:
```
t.add(1).cumprod()['2017'].plot()
```

```
/Users/janschneider/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launche
r.py:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a si
ngle string to slice the rows, like `frame[string]`, is deprecated and will be r
emoved in a future version. Use `frame.loc[string]` instead.
  """Entry point for launching an IPython kernel.
```

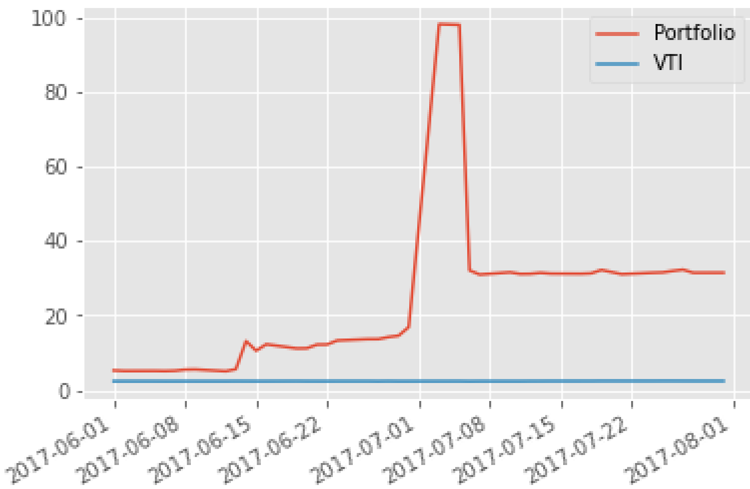
Out[24]:
```
<AxesSubplot:>
```



Zoom in a bit more:

In [27]:
```
t.add(1).cumprod()['2017-6':'2017-7'].plot()
```

Out[27]: <AxesSubplot:>



What stocks are in our portfolio in early July?

In [28]:
```
weights.loc['2017-7-5'].dropna()
```

Out[28]:
```
CAG      0.016792
VNRX     0.016375
YBAO     0.017201
CHRW     0.017221
ARAO     0.010321
LBTI     0.860048
ACFC     0.017355
ICCT     0.017201
LVBX     0.016674
SOWG     0.010812
Name: 2017-07-05 00:00:00, dtype: float64
```

Very large weight in LBTI!

Check LBTI price during this period:

In [29]:
```
PRICE.LBTI['2017-6-1':'2017-7-15']
```

Out[29]:
```
date
2017-06-01    0.000100
2017-06-02    0.000001
2017-06-05    0.000001
2017-06-06    0.000001
2017-06-07    0.000001
2017-06-08    0.000001
2017-06-09    0.000001
2017-06-12    0.000001
2017-06-13    0.000001
2017-06-14    0.000001
2017-06-15    0.000100
2017-06-16    0.000100
2017-06-19    0.000100
2017-06-20    0.000100
2017-06-21    0.000100
2017-06-22    0.000100
```

```
2017-06-23    0.000200
2017-06-26    0.000200
2017-06-27    0.000100
2017-06-28    0.000100
2017-06-29    0.000100
2017-06-30    0.000100
2017-07-03    0.005000
2017-07-05    0.005000
2017-07-06    0.001000
2017-07-07    0.001000
2017-07-10         NaN
2017-07-11         NaN
2017-07-12         NaN
2017-07-13    0.001000
2017-07-14         NaN
Name: LBTI, dtype: float64
```

Compare price and trading volume for this firm:

In [30]:

```
x = pd.DataFrame()
x['Price']   = PRICE        .LBTI['2017-6':'2017-7']
x['Volume']  = VOLUME       .LBTI['2017-6':'2017-7']
x['DVolume'] = DOLLAR_VOLUME.LBTI['2017-6':'2017-7']
x
```

Out[30]:

date	Price	Volume	DVolume
2017-06-01	0.000100	150.0	0.015000
2017-06-02	0.000001	931.0	0.000931
2017-06-05	0.000001	40.0	0.000040
2017-06-06	0.000001	0.0	0.000000
2017-06-07	0.000001	0.0	0.000000
2017-06-08	0.000001	0.0	0.000000
2017-06-09	0.000001	0.0	0.000000
2017-06-12	0.000001	0.0	0.000000
2017-06-13	0.000001	25.0	0.000025
2017-06-14	0.000001	0.0	0.000000
2017-06-15	0.000100	500.0	0.050000
2017-06-16	0.000100	0.0	0.000000
2017-06-19	0.000100	0.0	0.000000
2017-06-20	0.000100	0.0	0.000000
2017-06-21	0.000100	1096.0	0.109600
2017-06-22	0.000100	0.0	0.000000
2017-06-23	0.000200	240.0	0.048000
2017-06-26	0.000200	0.0	0.000000
2017-06-27	0.000100	502.0	0.050200

	Price	Volume	DVolume
date			
2017-06-28	0.000100	100.0	0.010000
2017-06-29	0.000100	0.0	0.000000
2017-06-30	0.000100	0.0	0.000000
2017-07-03	0.005000	10064.0	50.320000
2017-07-05	0.005000	0.0	0.000000
2017-07-06	0.001000	500.0	0.500000
2017-07-07	0.001000	0.0	0.000000
2017-07-10	NaN	NaN	NaN
2017-07-11	NaN	NaN	NaN
2017-07-12	NaN	NaN	NaN
2017-07-13	0.001000	13.0	0.013000
2017-07-14	NaN	NaN	NaN
2017-07-17	NaN	NaN	NaN
2017-07-18	0.001000	0.0	0.000000
2017-07-19	0.001000	14.0	0.014000
2017-07-20	0.001000	0.0	0.000000
2017-07-21	0.001000	0.0	0.000000
2017-07-24	0.001000	16909.0	16.909000
2017-07-25	0.001000	0.0	0.000000
2017-07-26	0.001000	44.0	0.044000
2017-07-27	0.001000	0.0	0.000000
2017-07-28	0.001000	0.0	0.000000
2017-07-31	0.001000	0.0	0.000000

So this is a "penny stock" with barely any volume.
Restrict firms to assets we can trade:

In [31]:
```
date = '2017-6-30'

p = PRICE[all_firms].loc[date]
p
```

Out[31]:
```
AIR     34.76
ABT     48.61
WDDD     0.03
ACU     28.60
AE      41.08
        ...
FHTX      NaN
GBNY      NaN
```

```
SPFR       NaN
PLTK       NaN
GEG        NaN
Name: 2017-06-30 00:00:00, Length: 5013, dtype: float64
```

In [32]:

```
p[p>1]   # All firms that have a price greater than $1 on this day
```

Out[32]:

```
AIR       34.76
ABT       48.61
ACU       28.60
AE        41.08
BKTI       3.75
          ...
XPER      29.80
ASRT      10.74
PRG       38.90
BNTC       1.85
TPL      293.78
Name: 2017-06-30 00:00:00, Length: 3265, dtype: float64
```

Put this filter into out select_asset function (and also add a filter for volume):

In [33]:

```
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 = SALESA[tradable_assets][:date].iloc[-1].nsmallest(10).index
    return assets


portfolio, weights, trades = run_backtest('quarter', '2010-1-1')

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

Out[33]: <AxesSubplot:>