Selecting Assets I: Price Data

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
```

Read the "close" table we downloaded from tiingo:

In [2]:

```
PRICE = pd.read_csv('data/tiingo/close.csv', index_col='date', parse_dates=['dat
PRICE
```

Out[2]:

	AIR	ABT	WDDD	ACU	AE	BKTI	AMD	APD	CECE	MATX	...	DMYI	A
date													
2009-04-15	14.43	42.66	0.1700	7.4300	14.65	0.6400	3.44	59.79	3.64	23.29	...	NaN	
2009-04-16	15.04	42.69	0.1700	7.6000	14.84	0.6300	3.57	60.48	3.88	24.52	...	NaN	
2009-04-17	15.17	43.89	0.1700	7.2000	15.00	0.5500	3.56	60.65	3.94	25.97	...	NaN	
2009-04-20	14.01	44.09	0.1700	7.3800	14.50	0.5598	3.31	57.79	3.79	23.24	...	NaN	
2009-04-21	14.66	44.37	0.1700	7.3900	14.50	0.5600	3.36	58.05	3.66	24.14	...	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	
2021-04-08	41.01	119.78	0.4100	38.4500	28.88	4.8900	83.35	283.11	8.20	72.13	...	11.00	
2021-04-09	40.89	120.90	0.3800	36.9200	27.71	4.8900	82.76	284.36	8.21	73.31	...	11.03	1
2021-04-12	40.92	121.04	0.3925	38.6900	26.00	4.6400	78.58	282.88	8.00	71.51	...	10.84	

	AIR	ABT	WDDD	ACU	AE	BKTI	AMD	APD	CECE	MATX	...	DMYI	A
date													
2021-04-13	40.23	123.01	0.3850	38.7501	26.34	4.5100	80.19	285.11	7.84	68.83	...	10.75	1
2021-04-14	41.08	121.50	0.3900	38.3057	26.59	4.6800	78.55	282.88	7.85	68.64	...	10.59	1

3021 rows × 5914 columns

Read the "adjClose" table and calculate the returns:

In [3]:

```
RET = pd.read_csv('data/tiingo/adjClose.csv', index_col='date', parse_dates=['da
RET
```

Out[3]:

	AIR	ABT	WDDD	ACU	AE	BKTI	AMD	APD
date								
2009-04-15	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2009-04-16	0.042273	0.000703	0.000000	0.022880	0.012969	-0.015625	0.037791	0.011540
2009-04-17	0.008644	0.028110	0.000000	-0.052632	0.010782	-0.126984	-0.002801	0.002811
2009-04-20	-0.076467	0.004557	0.000000	0.025000	-0.033333	0.017818	-0.070225	-0.047156
2009-04-21	0.046395	0.006351	0.000000	0.001355	0.000000	0.000357	0.015106	0.004499
...	...	...	...	...	...	...	...	...
2021-04-08	0.008608	-0.002747	0.000000	-0.035132	0.031429	-0.018072	0.013990	0.002195
2021-04-09	-0.002926	0.009350	-0.073171	-0.039792	-0.040512	0.004090	-0.007079	0.004415
2021-04-12	0.000734	0.001158	0.032895	0.047941	-0.061711	-0.051125	-0.050507	-0.005205
2021-04-13	-0.016862	0.016276	-0.019108	0.001553	0.013077	-0.028017	0.020489	0.007883
2021-04-14	0.021129	-0.008617	0.012987	-0.011468	0.009491	0.037694	-0.020451	-0.007822

3021 rows × 5914 columns

Get VTI as benchmark:

In [4]:

```
vti = tiingo.get_dataframe(['VTI'], '1999-1-1', metric_name='adjClose')
vti.index = pd.to_datetime(vti.index).tz_convert(None)
vti[:3]
```

Out[4]:

	VTI
2001-05-31	39.732523
2001-06-01	40.009405
2001-06-04	40.182456

Our backtest function:

In [5]:

```
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
```

In [6]:

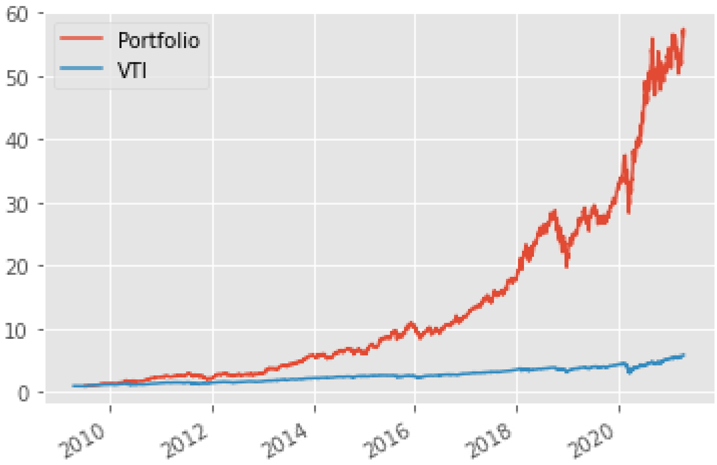
```
def select_assets(date):
    return ['AAPL','MSFT','AMZN','NFLX']


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


portfolio, weights, trades = run_backtest('quarter')
```

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

Out[6]: `<AxesSubplot:>`



Let's select firms by trading volume.

Read volume table:

In [7]:

```
VOLUME = pd.read_csv('data/tiingo/volume.csv', index_col='date', parse_dates=['d
VOLUME[-3:]
```

Out[7]:

	AIR	ABT	WDDD	ACU	AE	BKTI	AMD	APD	CECE	MATX	...
date											
2021-04-12	144326	2915975	29233.0	15972	22050	13762	62098804	682009	63625	239246	...
2021-04-13	257263	4944569	19659.0	6928	5979	6618	37767257	893679	45305	339259	...
2021-04-14	158866	4726734	23094.0	2274	11105	35817	34263832	631742	76101	391634	...

3 rows × 5914 columns

In [8]:

```
DOLLAR_VOLUME = VOLUME * PRICE
DOLLAR_VOLUME[-3:]
```

Out[8]:

	AIR	ABT	WDDD	ACU	AE	BKTI	AMI
date							
2021-04-12	5905819.92	3.529496e+08	11473.9525	617956.6800	573300.00	63855.68	4.879724e+0

date	AIR	ABT	WDDD	ACU	AE	BKTI	AMI
2021-04-13	10349690.49	6.082314e+08	7568.7150	268460.6928	157486.86	29847.18	3.028556e+0
2021-04-14	6526215.28	5.742982e+08	9006.6600	87107.1618	295281.95	167623.56	2.691424e+0

3 rows × 5914 columns

For example, select 10 firms with the highest most recent dollar volume:

In [9]:

```
DOLLAR_VOLUME.iloc[-1].nlargest(10)
```

Out[9]:

```
TSLA    3.551176e+10
AAPL    1.151602e+10
AMZN    1.048296e+10
MSFT    5.896701e+09
NVDA    5.889313e+09
FB      5.275546e+09
GME     3.520134e+09
GS      3.312169e+09
SQ      3.226481e+09
BA      3.196220e+09
Name: 2021-04-14 00:00:00, dtype: float64
```

Select 10 firms with the highest average dollar volume during the most recent 30 trading days:

In [10]:

```
DOLLAR_VOLUME[-30:].mean().nlargest(10)
```

Out[10]:

```
TSLA     2.624358e+10
AAPL     1.325759e+10
AMZN     1.062242e+10
MSFT     7.230169e+09
FB       5.954732e+09
BA       4.935624e+09
NVDA     4.647977e+09
GME      4.541444e+09
GOOGL    3.420479e+09
AMD      3.402152e+09
dtype: float64
```

Get the assets:

In [11]:

```
DOLLAR_VOLUME[-30:].mean().nlargest(10).index
```

Out[11]:

```
Index(['TSLA', 'AAPL', 'AMZN', 'MSFT', 'FB', 'BA', 'NVDA', 'GME', 'GOOGL',
       'AMD'],
      dtype='object')
```

Backtest this strategy:

In [13]:

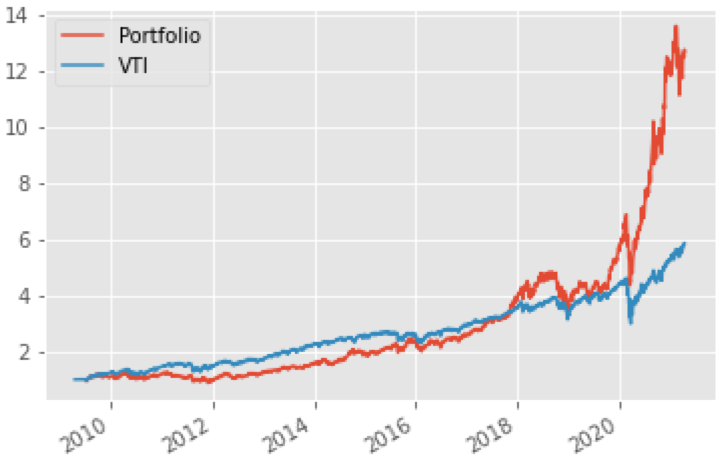
```
def select_assets(date):
    return DOLLAR_VOLUME[:date][-30:].mean().nlargest(10).index   # DOLLAR_VOLUME
```

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


portfolio, weights, trades = run_backtest('quarter')

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

```
Out[13]: <AxesSubplot:>
```



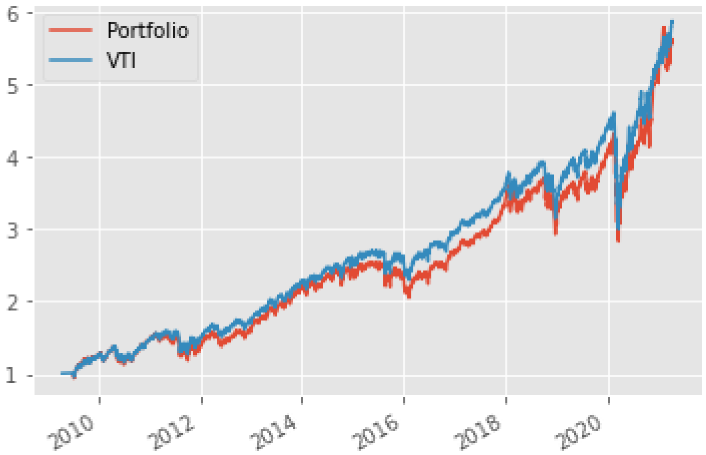
Top 100 most traded firms:

```
In [14]:
def select_assets(date):
    return DOLLAR_VOLUME[:date][-30:].mean().nlargest(100).index


portfolio, weights, trades = run_backtest('quarter')

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

```
Out[14]: <AxesSubplot:>
```


Portfolio
VTI