Selecting Assets II: Fundamental 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
```

Get data for revenues from SEC filings

Read file with SEC data:

In [2]:
```
pd.read_csv('data/sec/merged/2010q1.csv', parse_dates=['filed','ddate'])
```

Out[2]:

	cik	sic	countryinc	tag	filed	ddate
0	315189	3523	US	AccountsAndNotesReceivableNet	2010-03-01	2009-01-31
1	315189	3523	US	AccountsAndNotesReceivableNet	2010-03-01	2009-10-31
2	315189	3523	US	AccountsAndNotesReceivableNet	2010-03-01	2010-01-31
3	315189	3523	US	AccountsPayableAndAccruedLiabilitiesCurrentAnd...	2010-03-01	2009-01-31
4	315189	3523	US	AccountsPayableAndAccruedLiabilitiesCurrentAnd...	2010-03-01	2009-10-31
...	...	...	...	...	...	...
137522	1169055	1381	NaN	EntityCommonStockSharesOutstanding	2010-02-26	2010-01-31
137523	1169055	1381	NaN	EntityCommonStockSharesOutstanding	2010-02-26	2010-02-28

	cik	sic	countryinc	tag	filed	ddate
137524	1169055	1381	NaN	EntityPublicFloat	2010-02-26	2009-06-30
137525	1047862	4931	US	EntityCommonStockSharesOutstanding	2010-02-22	2010-01-31
137526	1047862	4931	US	EntityPublicFloat	2010-02-22	2009-06-30

137527 rows × 8 columns

Use this function to read this or more files and get values for a list of tags:

In [3]:
```
def get_items_from_SEC_files(tags, filename=None):                  # Function inp

    directory = 'data/sec/merged/'                                  # Read data fr
    filenames = [filename] if filename else os.listdir(directory) # Supplied fil
    filenames = [f for f in filenames if not f.startswith(".")]   # Exclude hidd

    results   = {t:pd.DataFrame() for t in tags}                    # Dictionary o

    for filename in sorted(filenames):                              # Loop over al
        print(filename)
        data = pd.read_csv(directory+filename, parse_dates=['filed','ddate'])  #

        for t in tags:                                              # Loop over al
            item  = data[data.tag==t]                               # Select all d
            short = item.sort_values(['cik','filed','ddate','qtrs'], ascending=[
            long  = item.sort_values(['cik','filed','ddate','qtrs'], ascending=[
            short = short.groupby(['cik','filed']).last()[['value','qtrs']]
            long  = long .groupby(['cik','filed']).last()[['value','qtrs']]
            short_long = short.join(long, lsuffix='_shortest', rsuffix='_longest
            results[t] = results[t].append( short_long )

    for t in tags:                                                  # Now sort all
        if not results[t].empty: results[t] = results[t].sort_index(level='filed

    return results
```

Run the function like this:

In [4]:
```
x = get_items_from_SEC_files(['Revenues','SalesRevenueGoodsNet'], '2010q1.csv')
```

```
2010q1.csv
```

The variable "x" is a dictionary of tables (one table for each tag):

In [5]:
```
x['Revenues']
```

Out[5]:

		value_shortest	qtrs_shortest	value_longest	qtrs_longest
cik	**filed**				

cik	filed	value_shortest	qtrs_shortest	value_longest	qtrs_longest
929887	**2010-01-07**	1.270301e+09	1	1.270301e+09	1
796343	**2010-01-22**	2.945853e+09	4	2.945853e+09	4
927653	**2010-01-26**	2.827200e+10	1	8.205900e+10	3
804328	**2010-01-27**	2.670000e+09	1	2.670000e+09	1
52988	**2010-01-28**	2.477785e+09	1	2.477785e+09	1
...	...	...	...	...	...
64978	**2010-03-30**	2.364320e+10	4	2.364320e+10	4
104169	**2010-03-30**	4.082140e+11	4	4.082140e+11	4
109198	**2010-03-30**	2.028844e+10	4	2.028844e+10	4
1058057	**2010-03-31**	2.807687e+09	4	2.807687e+09	4
1080224	**2010-03-31**	5.130000e+06	1	1.917400e+07	4

249 rows × 4 columns

All tags for Revenue (sales):

In [6]:
```
tags = ['RevenueFromContractWithCustomerExcludingAssessedTax','SalesRevenueNet',
        'SalesAndOtherOperatingRevenueIncludingSalesBasedTaxes','TotalRevenuesAn
```

Run the function for these tags:

In [7]:
```
items = get_items_from_SEC_files(tags)
```

```
2009q1.csv
2009q2.csv
2009q3.csv
2009q4.csv
2010q1.csv
2010q2.csv
2010q3.csv
2010q4.csv
2011q1.csv
2011q2.csv
2011q3.csv
2011q4.csv
2012q1.csv
2012q2.csv
2012q3.csv
2012q4.csv
2013q1.csv
2013q2.csv
2013q3.csv
2013q4.csv
2014q1.csv
2014q2.csv
2014q3.csv
2014q4.csv
2015q1.csv
2015q2.csv
```

```
2015q3.csv
2015q4.csv
2016q1.csv
2016q2.csv
2016q3.csv
2016q4.csv
2017q1.csv
2017q2.csv
2017q3.csv
2017q4.csv
2018q1.csv
2018q2.csv
2018q3.csv
2018q4.csv
2019q1.csv
2019q2.csv
2019q3.csv
2019q4.csv
2020_10.csv
2020_11.csv
2020_12.csv
2020q1.csv
2020q2.csv
2020q3.csv
2021_01.csv
2021_02.csv
2021_03.csv
```

Fix data error for "RevenueFromContractWithCustomerExcludingAssessedTax":

In [8]:
```
# RUN THIS CELL ONLY ONCE!
# Fixing data errors

# MKSI cik: 1049502
tag = 'RevenueFromContractWithCustomerExcludingAssessedTax'
t = items[tag].reset_index()
t.loc[(t.cik==1049502) & (t.filed=='2018-05-08'), 'value_longest']  /= 1000
t.loc[(t.cik==1049502) & (t.filed=='2018-05-08'), 'value_shortest'] /= 1000
items[tag] = t.set_index(['cik','filed'])
```

Check results for "Revenues":

In [9]:
```
items['Revenues']
```

Out[9]:

cik	filed	value_shortest	qtrs_shortest	value_longest	qtrs_longest
277948	**2009-04-15**	2.247000e+09	1	2.247000e+09	1
883984	**2009-04-23**	5.433500e+07	1	5.433500e+07	1
1365135	**2009-05-04**	1.201200e+09	1	1.201200e+09	1
901491	**2009-05-05**	2.849720e+08	1	2.849720e+08	1
1316631	**2009-05-05**	2.577800e+09	1	2.577800e+09	1
...	**...**	...	...	...	...
1769804	**2021-03-31**	1.648318e+07	4	1.648318e+07	4
1772720	**2021-03-31**	2.033800e+07	4	2.033800e+07	4

cik	filed	value_shortest	qtrs_shortest	value_longest	qtrs_longest
1776909	**2021-03-31**	3.962100e+07	4	3.962100e+07	4
1784970	**2021-03-31**	6.271564e+06	4	6.271564e+06	4
1813793	**2021-03-31**	3.500000e+02	4	3.500000e+02	4

115619 rows × 4 columns

For example, Apple "Revenues":

In [10]:
```
symbols = pd.read_csv('data/ticker_symbols/symbols.csv',index_col=0)      # read
symbols[:3]
```

Out[10]:

cik	ticker	title	exchange	assetType	priceCurrency	startDate	endDate
1750	AIR	AAR CORP	NYSE	Stock	USD	1984-07-19	2021-04-16
1800	ABT	ABBOTT LABORATORIES	NYSE	Stock	USD	1983-04-06	2021-04-16
1961	WDDD	WORLDS INC	OTCQB	Stock	USD	1998-10-20	2021-04-16

In [11]:
```
symbols[symbols.ticker=='AAPL']        # Find Apple CIK
```

Out[11]:

cik	ticker	title	exchange	assetType	priceCurrency	startDate	endDate
320193	AAPL	Apple Inc.	NASDAQ	Stock	USD	1980-12-12	2021-04-16

In [12]:
```
items['Revenues'].loc[320193]       # Apple revenues
```

Out[12]:

filed	value_shortest	qtrs_shortest	value_longest	qtrs_longest
2018-11-05	6.290000e+10	1	2.655950e+11	4

So Apple has only "Revenues" in 2018-11-5. Where are all the other ones?
Check all tags for Apple:

In [13]:
```
cik = symbols[symbols.ticker=='AAPL'].index[0]

t = [items[tag].loc[cik].value_shortest.rename(tag) for tag in tags if cik in it
pd.concat(t, 'columns')
```

Out[13]:

	RevenueFromContractWithCustomerExcludingAssessedTax	SalesRevenueNet	Revenues

filed	RevenueFromContractWithCustomerExcludingAssessedTax	SalesRevenueNet	Revenues
filed			
2009-07-22	NaN	8.337000e+09	NaN
2009-10-27	NaN	3.653700e+10	NaN
2010-01-25	NaN	1.568300e+10	NaN
2010-04-21	NaN	1.349900e+10	NaN
2010-07-21	NaN	1.570000e+10	NaN
2010-10-27	NaN	2.034300e+10	NaN
2011-01-19	NaN	2.674100e+10	NaN
2011-04-21	NaN	2.466700e+10	NaN
2011-07-20	NaN	2.857100e+10	NaN
2011-10-26	NaN	2.827000e+10	NaN
2012-01-25	NaN	4.633300e+10	NaN
2012-04-25	NaN	3.918600e+10	NaN
2012-07-25	NaN	3.502300e+10	NaN
2012-10-31	NaN	3.596600e+10	NaN
2013-01-24	NaN	5.451200e+10	NaN
2013-04-24	NaN	4.360300e+10	NaN
2013-07-24	NaN	3.532300e+10	NaN
2013-10-30	NaN	3.747200e+10	NaN
2014-01-28	NaN	5.759400e+10	NaN
2014-04-24	NaN	4.564600e+10	NaN
2014-07-23	NaN	3.743200e+10	NaN

filed	RevenueFromContractWithCustomerExcludingAssessedTax	SalesRevenueNet	Revenues
2014-10-27	NaN	4.212300e+10	NaN
2015-01-28	NaN	7.459900e+10	NaN
2015-04-28	NaN	5.801000e+10	NaN
2015-07-22	NaN	4.960500e+10	NaN
2015-10-28	NaN	5.150100e+10	NaN
2016-01-27	NaN	7.587200e+10	NaN
2016-04-27	NaN	5.055700e+10	NaN
2016-07-27	NaN	4.235800e+10	NaN
2016-10-26	NaN	4.685200e+10	NaN
2017-02-01	NaN	7.835100e+10	NaN
2017-05-03	NaN	5.289600e+10	NaN
2017-08-02	NaN	4.540800e+10	NaN
2017-11-03	NaN	5.257900e+10	NaN
2018-02-02	NaN	8.829300e+10	NaN
2018-05-02	NaN	6.113700e+10	NaN
2018-08-01	NaN	5.326500e+10	NaN
2018-11-05	NaN	NaN	6.290000e+10
2019-01-30	8.431000e+10	NaN	NaN
2019-05-01	5.801500e+10	NaN	NaN
2019-07-31	5.380900e+10	NaN	NaN
2019-10-31	6.404000e+10	NaN	NaN

filed	RevenueFromContractWithCustomerExcludingAssessedTax	SalesRevenueNet	Revenues
2020-01-29	9.181900e+10	NaN	NaN
2020-05-01	5.831300e+10	NaN	NaN
2020-07-31	5.968500e+10	NaN	NaN
2020-10-30	6.469800e+10	NaN	NaN
2021-01-28	1.114390e+11	NaN	NaN

Combine all all the tags to one "sales" item accoding to the order of this list:

In [14]:
```
tags
```

Out[14]:
```
['RevenueFromContractWithCustomerExcludingAssessedTax',
 'SalesRevenueNet',
 'Revenues',
 'SalesRevenueGoodsNet',
 'SalesAndOtherOperatingRevenueIncludingSalesBasedTaxes',
 'TotalRevenuesAndOtherIncome',
 'RevenuesNetOfInterestExpense']
```

(If a firm has reported values for multiple tags in this list, we use the first one.)

In [15]:
```
def combine_items(tags, items):
    result = items[tags[0]]
    for tag in tags[1:]:  result = result.combine_first( items[tag] )
    return result


items['sales'] = combine_items(tags, items)
```

Now check Apple sales:

In [16]:
```
items['sales'].loc[320193]
```

Out[16]:

filed	value_shortest	qtrs_shortest	value_longest	qtrs_longest
2009-07-22	8.337000e+09	1	2.666700e+10	3
2009-10-27	3.653700e+10	4	3.653700e+10	4
2010-01-25	1.568300e+10	1	1.568300e+10	1
2010-04-21	1.349900e+10	1	2.918200e+10	2
2010-07-21	1.570000e+10	1	4.488200e+10	3
2010-10-27	2.034300e+10	1	6.522500e+10	4

filed	value_shortest	qtrs_shortest	value_longest	qtrs_longest
2011-01-19	2.674100e+10	1	2.674100e+10	1
2011-04-21	2.466700e+10	1	5.140800e+10	2
2011-07-20	2.857100e+10	1	7.997900e+10	3
2011-10-26	2.827000e+10	1	1.082490e+11	4
2012-01-25	4.633300e+10	1	4.633300e+10	1
2012-04-25	3.918600e+10	1	8.551900e+10	2
2012-07-25	3.502300e+10	1	1.205420e+11	3
2012-10-31	3.596600e+10	1	1.565080e+11	4
2013-01-24	5.451200e+10	1	5.451200e+10	1
2013-04-24	4.360300e+10	1	9.811500e+10	2
2013-07-24	3.532300e+10	1	1.334380e+11	3
2013-10-30	3.747200e+10	1	1.709100e+11	4
2014-01-28	5.759400e+10	1	5.759400e+10	1
2014-04-24	4.564600e+10	1	1.032400e+11	2
2014-07-23	3.743200e+10	1	1.406720e+11	3
2014-10-27	4.212300e+10	1	1.827950e+11	4
2015-01-28	7.459900e+10	1	7.459900e+10	1
2015-04-28	5.801000e+10	1	1.326090e+11	2
2015-07-22	4.960500e+10	1	1.822140e+11	3
2015-10-28	5.150100e+10	1	2.337150e+11	4
2016-01-27	7.587200e+10	1	7.587200e+10	1
2016-04-27	5.055700e+10	1	1.264290e+11	2
2016-07-27	4.235800e+10	1	1.687870e+11	3
2016-10-26	4.685200e+10	1	2.156390e+11	4
2017-02-01	7.835100e+10	1	7.835100e+10	1
2017-05-03	5.289600e+10	1	1.312470e+11	2
2017-08-02	4.540800e+10	1	1.766550e+11	3
2017-11-03	5.257900e+10	1	2.292340e+11	4
2018-02-02	8.829300e+10	1	8.829300e+10	1
2018-05-02	6.113700e+10	1	1.494300e+11	2
2018-08-01	5.326500e+10	1	2.026950e+11	3
2018-11-05	6.290000e+10	1	2.655950e+11	4
2019-01-30	8.431000e+10	1	8.431000e+10	1
2019-05-01	5.801500e+10	1	1.423250e+11	2

filed	value_shortest	qtrs_shortest	value_longest	qtrs_longest
2019-07-31	5.380900e+10	1	1.961340e+11	3
2019-10-31	6.404000e+10	1	2.601740e+11	4
2020-01-29	9.181900e+10	1	9.181900e+10	1
2020-05-01	5.831300e+10	1	1.501320e+11	2
2020-07-31	5.968500e+10	1	2.098170e+11	3
2020-10-30	6.469800e+10	1	2.745150e+11	4
2021-01-28	1.114390e+11	1	1.114390e+11	1

Calculate the quarterly and annual values:

In [17]:
```
def calculate_quarterly_annual_values(item):                          # item: tabl
    result           = pd.DataFrame()                                 # Results go
    all_firms        = item.index.get_level_values('cik').unique()    # All CIKs.
    all_filing_dates = pd.read_csv('data/sec/dates/filing_dates.csv', index_col=

    for cik in all_firms:                                             # Loop over
        filing_dates = pd.Series(all_filing_dates.filed[cik])         # All filing

        # Quarterly values:
        valuesQ = item.loc[cik].value_shortest.reindex(filing_dates) # Values wi
        qtrsQ   = item.loc[cik].qtrs_shortest.astype(int)             # Number of
        for date,q in qtrsQ[qtrsQ>1].iteritems():                     # Loop over
            previous_values = valuesQ[:date][-q:-1]                   # Example: f
            if len(previous_values) == q-1:                           # If all pre
                valuesQ[date] -= previous_values.sum(skipna=False)    # Subtract p
            else:
                valuesQ[date]  = np.nan

        # Annual values:
        valuesA = item.loc[cik].value_longest.reindex(filing_dates) # Values wit
        qtrsA   = item.loc[cik].qtrs_longest.astype(int)            # Number of
        for date,q in qtrsA[qtrsA<4].iteritems():                   # Loop over
            previous_values = valuesQ[:date][-4:-q]                 # Example: f
            if len(previous_values) == 4-q:                         # If all pre
                valuesA[date] += previous_values.sum(skipna=False)  # Add previo
            else:
                valuesA[date]  = np.nan

        result = result.append( pd.DataFrame({'cik':cik, 'filed':filing_dates, '

    return result.set_index(['cik','filed'])                        # Return a t



sales = calculate_quarterly_annual_values(items['sales'])
```

Apple:

In [18]:
```
sales.loc[320193]
```

Out[18]:

filed	valueQ	valueA
2009-07-22	8.337000e+09	NaN
2009-10-27	NaN	3.653700e+10
2010-01-25	1.568300e+10	NaN
2010-04-21	1.349900e+10	NaN
2010-07-21	1.570000e+10	NaN
2010-10-27	2.034300e+10	6.522500e+10
2011-01-19	2.674100e+10	7.628300e+10
2011-04-21	2.466700e+10	8.745100e+10
2011-07-20	2.857100e+10	1.003220e+11
2011-10-26	2.827000e+10	1.082490e+11
2012-01-25	4.633300e+10	1.278410e+11
2012-04-25	3.918600e+10	1.423600e+11
2012-07-25	3.502300e+10	1.488120e+11
2012-10-31	3.596600e+10	1.565080e+11
2013-01-24	5.451200e+10	1.646870e+11
2013-04-24	4.360300e+10	1.691040e+11
2013-07-24	3.532300e+10	1.694040e+11
2013-10-30	3.747200e+10	1.709100e+11
2014-01-28	5.759400e+10	1.739920e+11
2014-04-24	4.564600e+10	1.760350e+11
2014-07-23	3.743200e+10	1.781440e+11
2014-10-27	4.212300e+10	1.827950e+11
2015-01-28	7.459900e+10	1.998000e+11
2015-04-28	5.801000e+10	2.121640e+11
2015-07-22	4.960500e+10	2.243370e+11
2015-10-28	5.150100e+10	2.337150e+11
2016-01-27	7.587200e+10	2.349880e+11
2016-04-27	5.055700e+10	2.275350e+11
2016-07-27	4.235800e+10	2.202880e+11
2016-10-26	4.685200e+10	2.156390e+11
2017-02-01	7.835100e+10	2.181180e+11
2017-05-03	5.289600e+10	2.204570e+11
2017-08-02	4.540800e+10	2.235070e+11
2017-11-03	5.257900e+10	2.292340e+11

filed	valueQ	valueA
2018-02-02	8.829300e+10	2.391760e+11
2018-05-02	6.113700e+10	2.474170e+11
2018-08-01	5.326500e+10	2.552740e+11
2018-11-05	6.290000e+10	2.655950e+11
2019-01-30	8.431000e+10	2.616120e+11
2019-05-01	5.801500e+10	2.584900e+11
2019-07-31	5.380900e+10	2.590340e+11
2019-10-31	6.404000e+10	2.601740e+11
2020-01-29	9.181900e+10	2.676830e+11
2020-05-01	5.831300e+10	2.679810e+11
2020-07-31	5.968500e+10	2.738570e+11
2020-10-30	6.469800e+10	2.745150e+11
2021-01-28	1.114390e+11	2.941350e+11

Save the "sales" table:

In [19]:
```
sales.to_csv('data/sec/items/Sales.csv')
```

And now we can read the file like this:

In [20]:
```
sales = pd.read_csv('data/sec/items/Sales.csv',  parse_dates=['filed'], index_co
```

We want to generate a table with assets in columns and trading days in rows (to use inside our "select_asset" function).
Get trading days:

In [21]:
```
trading_days = pd.to_datetime( tiingo.get_dataframe('SPY','2009-04-15').index ).
trading_days
```

Out[21]:
```
DatetimeIndex(['2009-04-15', '2009-04-16', '2009-04-17', '2009-04-20',
               '2009-04-21', '2009-04-22', '2009-04-23', '2009-04-24',
               '2009-04-27', '2009-04-28',
               ...
               '2021-04-09', '2021-04-12', '2021-04-13', '2021-04-14',
               '2021-04-15', '2021-04-16', '2021-04-19', '2021-04-20',
               '2021-04-21', '2021-04-22'],
              dtype='datetime64[ns]', name='date', length=3027, freq=None)
```

Now forward-fill the sales data to all trading days:

In [22]:
```
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




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

salesA[-3:]
```

Out[22]:

cik	1750	1800	1961	2034	2098	2178	2186	
date								
2021-04-20	1.748300e+09	3.460800e+10	199.0	NaN	164003040.0	1.022422e+09	44139000.0	9.763(
2021-04-21	1.748300e+09	3.460800e+10	199.0	NaN	164003040.0	1.022422e+09	44139000.0	9.763(
2021-04-22	1.748300e+09	3.460800e+10	199.0	NaN	164003040.0	1.022422e+09	44139000.0	9.763(

3 rows × 10144 columns

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

In [23]:
```
symbols.ticker
```

Out[23]:
```
cik
1750         AIR
1800         ABT
1961        WDDD
2098         ACU
2178          AE
           ...
1852736     TRIS
1852931    NHLDW
1853404     CEAS
1854074      LPC
1854445     VRAR
Name: ticker, Length: 8661, dtype: object
```

In [30]:
```
SALESQ = salesQ.rename(columns=symbols.ticker)
SALESA = salesA.rename(columns=symbols.ticker)

SALESA[-3:]
```

Out[30]:

cik date	AIR	ABT	WDDD	2034	ACU	AE	BKTI	
2021-04-20	1.748300e+09	3.460800e+10	199.0	NaN	164003040.0	1.022422e+09	44139000.0	9.763
2021-04-21	1.748300e+09	3.460800e+10	199.0	NaN	164003040.0	1.022422e+09	44139000.0	9.763
2021-04-22	1.748300e+09	3.460800e+10	199.0	NaN	164003040.0	1.022422e+09	44139000.0	9.763

3 rows × 10144 columns

Select firms based on sales

In [31]:

```
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 price data:

In [32]:
```
PRICE = pd.read_csv('data/tiingo/close.csv', index_col='date', parse_dates=['dat
```

In [33]:
```
RET = pd.read_csv('data/tiingo/adjClose.csv', index_col='date', parse_dates=['da
```

Get benchmark:

In [34]:
```
vti = tiingo.get_dataframe(['VTI'], '1990-1-1', metric_name='adjClose')
vti.index = pd.to_datetime(vti.index).tz_convert(None)
```

Top 10 firms with highest most recent sales:

In [35]:
```
SALESA.iloc[-1].nlargest(10)
```

Out[35]:
```
cik
WMT       5.552330e+11
AMZN      3.860640e+11
AAPL      2.941350e+11
CVS       2.687060e+11
UNH       2.571410e+11
MCK       2.376210e+11
ABC       1.945457e+11
GOOGL     1.825270e+11
COST      1.786260e+11
XOM       1.785740e+11
Name: 2021-04-22 00:00:00, dtype: float64
```

Suppose we select firms like this at the end of each quarter:

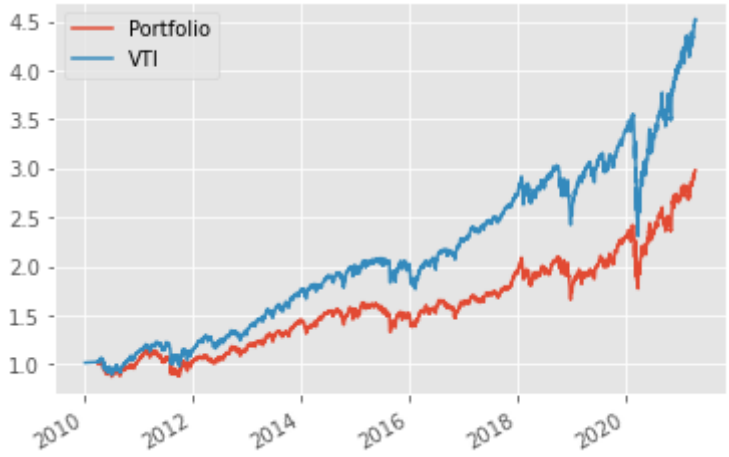
In [36]:
```
def select_assets(date):
    assets = SALESA[:date].iloc[-1].nlargest(10).index
    return assets


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[36]:
```
<AxesSubplot:>
```


Portfolio
VTI
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
2010
2012
2014
2016
2018
2020