CAPM - Capital Asset Pricing Model

Watch the video for the full overview.

Portfolio Returns:

$$r_p(t) = \sum_i^n w_i r_i(t)$$

Market Weights:

$$w_i = \frac{MarketCap_i}{\sum_j^n MarketCap_j}$$

CAPM of a portfolio

$$r_p(t) = \beta_p r_m(t) + \sum_i^n w_i \alpha_i(t)$$

In [1]:
```
# Model CAPM as a simple linear regression
```

In [20]:
```
from scipy import stats
```

In [3]:
```
help(stats.linregress)
```

```
Help on function linregress in module scipy.stats._stats_mstats_common:

linregress(x, y=None)
    Calculate a linear least-squares regression for two sets of measurements.

    Parameters
    ----------
    x, y : array_like
        Two sets of measurements.  Both arrays should have the same length.
        If only x is given (and y=None), then it must be a two-dimensional
        array where one dimension has length 2.  The two sets of measurements
        are then found by splitting the array along the length-2 dimension.

    Returns
    -------
    slope : float
        slope of the regression line
    intercept : float
        intercept of the regression line
    rvalue : float
        correlation coefficient
    pvalue : float
        two-sided p-value for a hypothesis test whose null hypothesis is
        that the slope is zero.
    stderr : float
        Standard error of the estimated gradient.

    See also
    --------
    :func:`scipy.optimize.curve_fit` : Use non-linear
     least squares to fit a function to data.
    :func:`scipy.optimize.leastsq` : Minimize the sum of
     squares of a set of equations.

    Examples
    --------
    >>> import matplotlib.pyplot as plt
    >>> from scipy import stats
    >>> np.random.seed(12345678)
    >>> x = np.random.random(10)
    >>> y = np.random.random(10)
    >>> slope, intercept, r_value, p_value, std_err = stats.linregress(x, y)

    To get coefficient of determination (r_squared)

    >>> print("r-squared:", r_value**2)
    ('r-squared:', 0.080402268539028335)

    Plot the data along with the fitted line

    >>> plt.plot(x, y, 'o', label='original data')
    >>> plt.plot(x, intercept + slope*x, 'r', label='fitted line')
    >>> plt.legend()
    >>> plt.show()
```

In [7]:
```
import pandas as pd
```

In [13]:
```
#import pandas_datareader as web
```

In [3]:
```
#spy_etf = web.DataReader('SPY','google')
import yfinance as yf
spy_etf = yf.download("SPY", start="2010-01-04", end="2017-07-18")
```

```
[*********************100%***********************]  1 of 1 completed
```

In [4]:
```
spy_etf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1897 entries, 2010-01-04 to 2017-07-17
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Open       1897 non-null   float64
 1   High       1897 non-null   float64
 2   Low        1897 non-null   float64
 3   Close      1897 non-null   float64
 4   Adj Close  1897 non-null   float64
 5   Volume     1897 non-null   int64  
dtypes: float64(5), int64(1)
memory usage: 103.7 KB
```

In [5]:
```
spy_etf.head()
```

Out[5]:

Date	Open	High	Low	Close	Adj Close	Volume
2010-01-04	112.370003	113.389999	111.510002	113.330002	88.860367	118944600
2010-01-05	113.260002	113.680000	112.849998	113.629997	89.095604	111579900
2010-01-06	113.519997	113.989998	113.430000	113.709999	89.158325	116074400
2010-01-07	113.500000	114.330002	113.180000	114.190002	89.534691	131091100
2010-01-08	113.889999	114.620003	113.660004	114.570000	89.832649	126402800

In [8]:
```
start = pd.to_datetime('2010-01-04')
end = pd.to_datetime('2017-07-18')
```

In [9]:
```
#aapl = web.DataReader('AAPL','google',start,end)
aapl = yf.download("AAPL", start="2010-01-04", end="2017-07-18")
```

```
[*********************100%***********************]  1 of 1 completed
```

In [10]:
```
aapl.head()
```

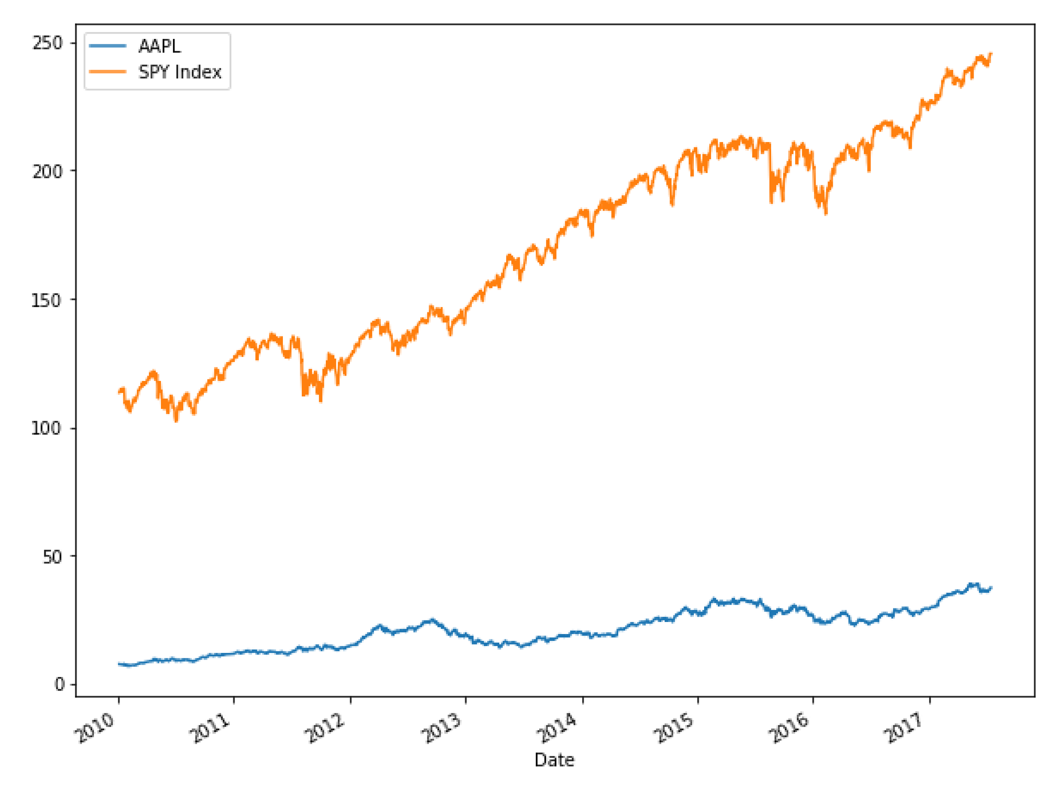
Out[10]:

Date	Open	High	Low	Close	Adj Close	Volume
2010-01-04	7.622500	7.660714	7.585000	7.643214	6.526019	493729600
2010-01-05	7.664286	7.699643	7.616071	7.656429	6.537303	601904800
2010-01-06	7.656429	7.686786	7.526786	7.534643	6.433318	552160000
2010-01-07	7.562500	7.571429	7.466071	7.520714	6.421427	477131200
2010-01-08	7.510714	7.571429	7.466429	7.570714	6.464116	447610800

In [11]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [12]:
```
aapl['Close'].plot(label='AAPL',figsize=(10,8))
spy_etf['Close'].plot(label='SPY Index')
plt.legend()
```

Out[12]:
```
<matplotlib.legend.Legend at 0xd8ba52f0a0>
```

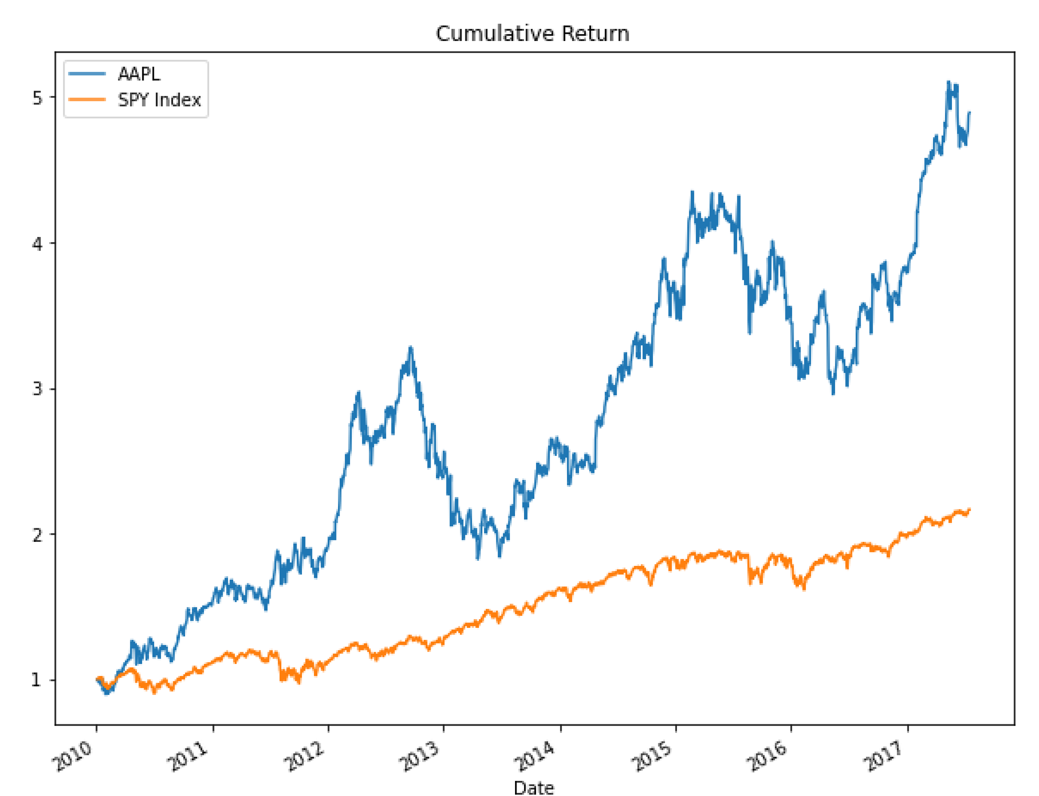


Compare Cumulative Return

In [13]:
```
aapl['Cumulative'] = aapl['Close']/aapl['Close'].iloc[0]
spy_etf['Cumulative'] = spy_etf['Close']/spy_etf['Close'].iloc[0]
```

In [14]:
```
aapl['Cumulative'].plot(label='AAPL',figsize=(10,8))
spy_etf['Cumulative'].plot(label='SPY Index')
plt.legend()
plt.title('Cumulative Return')
```

Out[14]:
```
Text(0.5, 1.0, 'Cumulative Return')
```

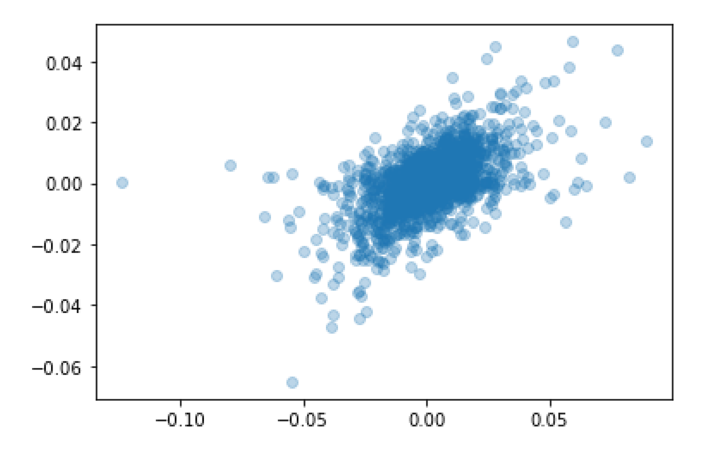


Get Daily Return

In [15]:
```
aapl['Daily Return'] = aapl['Close'].pct_change(1)
spy_etf['Daily Return'] = spy_etf['Close'].pct_change(1)
```

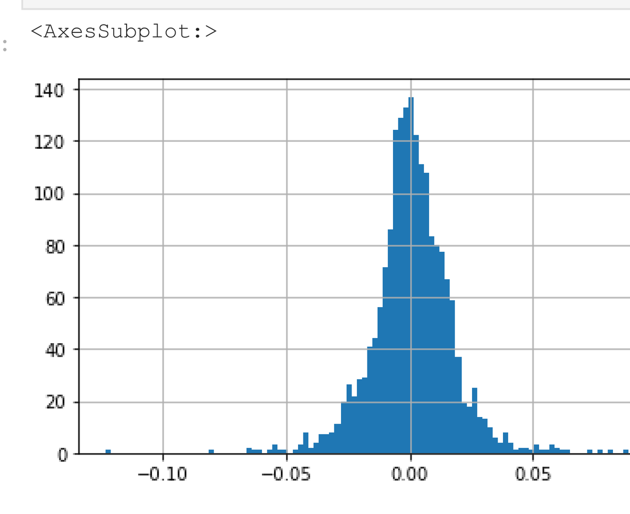
In [16]:
```
plt.scatter(aapl['Daily Return'],spy_etf['Daily Return'],alpha=0.3)
```

Out[16]:
```
<matplotlib.collections.PathCollection at 0xd8b4db2940>
```



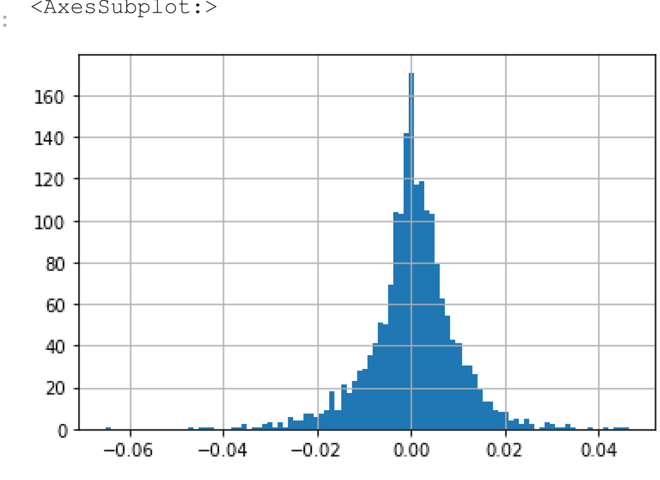
In [17]:
```
aapl['Daily Return'].hist(bins=100)
```

Out[17]:
```
<AxesSubplot:>
```



In [18]:
```
spy_etf['Daily Return'].hist(bins=100)
```

Out[18]:
```
<AxesSubplot:>
```



In [21]:
```
beta,alpha,r_value,p_value,std_err = stats.linregress(aapl['Daily Return'].iloc[1:],spy_etf['Daily Return'].ilo
```

In [22]:
```
beta
```

Out[22]:
```
0.32572226197243687
```

In [23]:
```
alpha
```

Out[23]:
```
0.0001373797735205996
```

In [40]:
```
r_value
```

Out[40]:
```
0.33143080741409325
```

What if our stock was completely related to SP500?

In [24]:
```
spy_etf['Daily Return'].head()
```

Out[24]:
```
Date
2010-01-04         NaN
2010-01-05    0.002647
2010-01-06    0.000704
2010-01-07    0.004221
2010-01-08    0.003328
Name: Daily Return, dtype: float64
```

In [50]:
```
import numpy as np
```

In [63]:
```
noise = np.random.normal(0,0.001,len(spy_etf['Daily Return'].iloc[1:]))
```

In [64]:
```
noise
```

Out[64]:
```
array([ 0.00089285,  0.00056301, -0.00022182, ..., -0.00075069,
       -0.00017751, -0.00034175])
```

In [65]:
```
spy_etf['Daily Return'].iloc[1:] + noise
```

Out[65]:
```
Date
2010-01-05    0.003540
2010-01-06    0.001267
2010-01-07    0.003999
2010-01-08    0.003744
2010-01-11    0.002430
2010-01-12   -0.008178
2010-01-13    0.006244
2010-01-14    0.003002
2010-01-15   -0.010593
2010-01-19    0.012289
2010-01-20   -0.010683
2010-01-21   -0.019239
2010-01-22   -0.022449
2010-01-25    0.003428
2010-01-26   -0.003510
2010-01-27    0.004741
2010-01-28   -0.011874
2010-01-29   -0.010649
2010-02-01    0.015995
2010-02-02    0.011317
2010-02-03   -0.006852
2010-02-04   -0.029828
2010-02-05    0.000845
2010-02-08   -0.007362
2010-02-09    0.010399
2010-02-10   -0.000745
2010-02-11    0.008745
2010-02-12   -0.001782
2010-02-16    0.016414
2010-02-17    0.003658
                 ...   
2017-06-06   -0.001963
2017-06-07    0.003158
2017-06-08   -0.000087
2017-06-09   -0.000582
2017-06-12   -0.001564
2017-06-13    0.006079
2017-06-14   -0.002911
2017-06-15   -0.001163
2017-06-16   -0.004836
2017-06-19    0.008102
2017-06-20   -0.008464
2017-06-21   -0.000669
2017-06-22   -0.000532
2017-06-23    0.002970
2017-06-26    0.001098
2017-06-27   -0.009717
2017-06-28    0.008961
2017-06-29   -0.010516
2017-06-30    0.001965
2017-07-03    0.001218
2017-07-05    0.003210
2017-07-06   -0.008497
2017-07-07    0.005798
2017-07-10    0.000335
2017-07-11   -0.001653
2017-07-12    0.008218
2017-07-13    0.003311
2017-07-14    0.003913
2017-07-17   -0.000300
2017-07-18    0.000188
Name: Daily Return, Length: 1896, dtype: float64
```

In [66]:
```
beta,alpha,r_value,p_value,std_err = stats.linregress(spy_etf['Daily Return'].iloc[1:]+noise,spy_etf['Daily Ret
```

In [67]:
```
beta
```

Out[67]:
```
0.9902145894162997
```

In [68]:
```
alpha
```

Out[68]:
```
1.4861814214210613e-05
```

Looks like our understanding is correct!