Imports & Settings

In [1]:

```
import json
import numpy as np
import pandas as pd
import yfinance as yf
from collections import Counter

# Visualization tools
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

CIKs for S&P 500 Constituents

In [3]:

```
df = pd.read_csv('../data/sp500_ciks.csv', dtype=str)
df.head()
```

Out[3]:

	Symbol	Security	SEC filings	GICS Sector	GICS Sub-Industry	Headquarters Location	Date first added	CIK	Founded
0	MMM	3M	reports	Industrials	Industrial Conglomerates	Saint Paul, Minnesota	1976-08-09	0000066740	1902
1	ABT	Abbott Laboratories	reports	Health Care	Health Care Equipment	North Chicago, Illinois	1964-03-31	0000001800	1888
2	ABBV	AbbVie	reports	Health Care	Pharmaceuticals	North Chicago, Illinois	2012-12-31	0001551152	2013 (1888)
3	ABMD	Abiomed	reports	Health Care	Health Care Equipment	Danvers, Massachusetts	2018-05-31	0000815094	1981
4	ACN	Accenture	reports	Information Technology	IT Consulting & Other Services	Dublin, Ireland	2011-07-06	0001467373	1989

In [3]:

```
df.shape
```

Out[3]:

```
(505, 9)
```

Plotting a Concept for a Given Company

In [4]:

```
def get_company_cik(ticker):
    '''
    Description
    -----------
    Returns the CIK value for a company given their ticker.

    Parameters
    -----------
    ticker : str
        A company's ticker symbol for their publicly traded stock.

    Example
    -----------
    >>> get_company_cik('MSFT')
    '0000789019'
    '''
    return df.loc[df.Symbol == ticker, 'CIK'].values[0]
```

In [11]:

```
def get_company_facts(ticker):
    '''
    Description
    -----------
    Returns the company facts for a given ticker from the appropriate
    file downloaded through the SEC's bulk data archive in JSON format.

    Parameters
    -----------
    ticker : str
        A company's ticker symbol for their publicly traded stock.

    Example
    -----------
    >>> get_company_facts('MSFT')
    {'cik': 789019,
     'entityName': 'MICROSOFT CORPORATION',
     'facts': {'dei': {'EntityCommonStockSharesOutstanding': {
     'label': 'Entity Common Stock, Shares Outstanding',
      'description': "Indicate number of shares or other units
      outstanding of each of registrant's classes of capital or
      common stock or other ownership interests, if...
    '''
    cik = get_company_cik(ticker)
    with open(f'../data/sec_bulk_data/CIK{cik}.json') as f:
        facts = json.load(f)
    return facts
```

In [6]:

```
def get_company_concepts(ticker):
    '''
    Description
    -----------
    Returns a list of the available concepts for a given ticker.

    Parameters
    -----------
    ticker : str
        A company's ticker symbol for their publicly traded stock.

    Example
    -----------
    >>> get_company_concepts('MSFT')
    ['AccountsPayableCurrent',
     'AccountsReceivableNet',
     'AccountsReceivableNetCurrent',
     'AccountsReceivableNetNoncurrent',
     'AccruedIncomeTaxesCurrent',
     ...]
    '''
    facts = get_company_facts(ticker)
    return list(facts['facts']['us-gaap'].keys())
```

In [7]:

```
def get_concept_description(ticker, concept, label=False):
    '''
    Description
    -----------
    Returns the description for a given concept for a given
    ticker. Optionally returns the label of the concept.

    Parameters
    -----------
    ticker : str
        A company's ticker symbol for their publicly traded stock.

    concept : str
        The name of the concept. Must be available for the
        specified ticker. See `get_company_concepts()`.

    label : bool
        Whether to return the label for the concept as well.

    Example
    -----------
    >>> get_concept_description('MSFT', 'Assets', label=True)
    ('Assets',
     'Sum of the carrying amounts as of the balance sheet date
     of all assets that are recognized. Assets are probable future
     economic benefits obtained or controlled by an entity as a
     result of past transactions or events.')
    '''
    facts = get_company_facts(ticker)
    concept = facts['facts']['us-gaap'][concept]
    try:
        if label:
            return concept['label'], concept['description']
        return concept['description']
    except KeyError as e:
        print(f'KeyError: The specified company does not have a concept of \'{concept}\'')
```

In [34]:

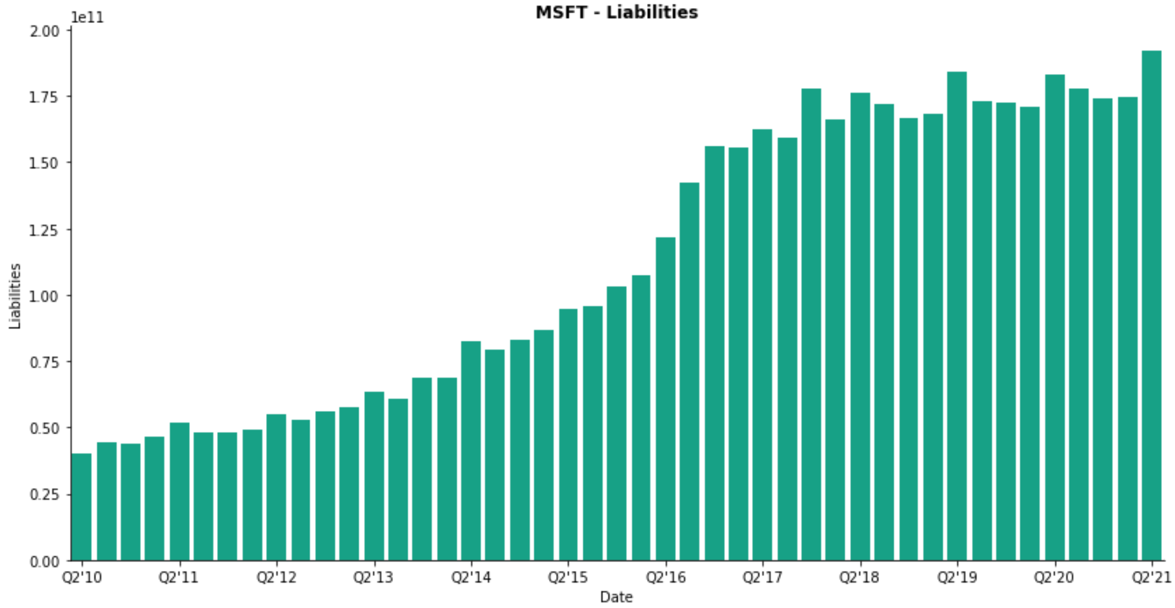
```python
def get_concept_values(ticker, concept, frame_only=True, frame_filter='Q', as_df=False):
    '''
    Description
    -----------
    Returns the values for a given concept for a given ticker.

    Parameters
    -----------
    ticker : str
        A company's ticker symbol for their publicly traded stock.

    concept : str
        The name of the concept. Must be available for the
        specified ticker. See `get_company_concepts()`.

    frame_only : bool, default=True
        Whether to filter the concept values to only those with
        specified frames. A frame aligns a company report with
        the nearest calendar quarter or year and is useful for
        comparison purposes.

    frame_filter : str, default='Q'
        Only applicable when `frame_only=True`. Filters the frames
        to the specified frequency (e.g., 'Q' for quarterly or 'Y'
        for yearly).

    as_df : bool, default=False
        Whether to return the data as a pandas.DataFrame() object.
        Note: This will only return the date of observation and the
        actual concept values while ignoring the other information.

    Example
    -----------
    >>> df = get_concept_values('MSFT', 'Assets', as_df=True)
    >>> df.head()
                Assets
    2009-06-30  77888000000
    2009-09-30  81612000000
    2009-12-31  82096000000
    2010-03-31  84910000000
    2010-06-30  86113000000
    '''
    facts = get_company_facts(ticker)
    concept_values = facts['facts']['us-gaap'][concept]['units']['USD']

    if frame_only:
        concept_values = [item for item in concept_values if 'frame' in item.keys()]
        concept_values = [item for item in concept_values if frame_filter in item['frame']]

    if as_df:
        df = pd.DataFrame(data={concept: [item['val'] for item in concept_values]},
                          index=[pd.to_datetime(item['end']) for item in concept_values])
        df.sort_index(ascending=True, inplace=True)
        return df

    return concept_values
```

In [9]:

```
def plot_concept(ticker, concept):
    '''
    Description
    -----------
    Plots a bar chart of a given concept for a given ticker.
    Useful for quickly visualizing trends in concepts.

    Parameters
    -----------
    ticker : str
        A company's ticker symbol for their publicly traded stock.

    concept : str
        The name of the concept. Must be available for the
        specified ticker. See `get_company_concepts()`.

    Example
    -----------
    >>> plot_concept('MSFT', 'Assets')
    <matplotlib.axes._subplots.AxesSubplot>
    '''
    # Storing the data in a dataframe for easier plotting
    concept_values = get_concept_values(ticker, concept)
    df = pd.DataFrame({'Date': [pd.to_datetime(item['end']) for item in concept_values],
                       'Frame': [item['frame'] for item in concept_values],
                       concept: [item['val'] for item in concept_values]})

    # Changing format of the Frame column
    def frame_format(x):
        q = x[x.index('Q'):x.index('Q')+2]
        y = x[4:6]
        return f"{q}'{y}"
    df.Frame = df.Frame.apply(frame_format)

    # Plotting
    sns.barplot(data=df, x='Date', y=concept, color='#00b894')
    sns.despine()
    plt.title(f'{ticker} - {concept}', fontweight='bold')
    plt.xticks(ticks=np.arange(0, len(df), 4), labels=df.Frame[::4])
```

In [12]:

```
fig = plt.figure(figsize=(14, 7))
plot_concept('MSFT', 'Liabilities')
```



Retrieving Price Histories

Downloading Data

In [11]:

```
data = yf.download(tickers=df.Symbol.values.tolist(),
                   period='max',
                   group_by='ticker')
```

```
[*********************100%***********************]  505 of 505 completed

1 Failed download:
- BF.B: 1d data not available for startTime=-2208988800 and endTime=1629247883. Only 100 years worth of day granularity data are allo
wed to be fetched per request.
```

It appears that the ticker `BF.B` failed. The '.B' portion of the ticker refers to share class. Referencing the share class in a ticker is not fully standardized across different financial sources and can sometimes be represented as a hyphen instead of a period. I will try to download the data for `BF.B` by manually changing it to `BF-B` instead.

In [12]:

```
bfb = yf.download(tickers='BF-B', period='max')
bfb.head()
```

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

Out[12]:

	Open	High	Low	Close	Adj Close	Volume
Date						
1980-03-17	0.0	0.447407	0.442963	0.442963	0.204822	202500
1980-03-18	0.0	0.442963	0.442963	0.442963	0.204822	84375
1980-03-19	0.0	0.448889	0.442963	0.448889	0.207562	329063
1980-03-20	0.0	0.450370	0.447407	0.450370	0.208247	354375
1980-03-21	0.0	0.450370	0.438519	0.444444	0.205507	2910938

In [13]:

```
data.shape
```

Out[13]:

```
(15011, 3030)
```

In [14]:

```
data.columns
```

Out[14]:

```
MultiIndex([('MRNA',      'Open'),
            ('MRNA',      'High'),
            ('MRNA',       'Low'),
            ('MRNA',     'Close'),
            ('MRNA', 'Adj Close'),
            ('MRNA',    'Volume'),
            ( 'VTR',      'Open'),
            ( 'VTR',      'High'),
            ( 'VTR',       'Low'),
            ( 'VTR',     'Close'),
            ...
            ( 'SYK',       'Low'),
            ( 'SYK',     'Close'),
            ( 'SYK', 'Adj Close'),
            ( 'SYK',    'Volume'),
            ('FITB',      'Open'),
            ('FITB',      'High'),
            ('FITB',       'Low'),
            ('FITB',     'Close'),
            ('FITB', 'Adj Close'),
            ('FITB',    'Volume')],
           length=3030)
```

Exporting Individual Histories

The data returned by the `yfinance.download()` function placed everything into a single dataframe with columns grouped by ticker. While useful for quickly analyzing / plotting any of the companies' histories, there are two primary issues with having everything in one dataframe:

- The file is much larger than the 100MB limit imposed by GitHub. While solutions for handling large file uploads exist, it's preferable to avoid such workarounds unless absolutely needed.
- Thinking forward to the interactive dashboard portion of this project, loading in all of the price history data for every company in the S&P 500 whenever we want to analyze just one is a waste of resources and will likely lead to unwanted load times.

By splitting the data up into individual files, both of these issues are resolved. To start, I'll export the `BF.B` / `BF-B` dataframe and then move on to looping over all of the others.

In [15]:

```
bfb.to_csv('../data/price_histories/BF-B_history.csv')
print('Successfully saved BF-B\'s history.')
```

```
Successfully saved BF-B's history.
```

In [18]:

```
tickers = df.Symbol.values.tolist()
```

In [16]:

```
for i, ticker in enumerate(tickers):
    if ticker == 'BF.B':
        continue
    df_temp = data[ticker].dropna()
    df_temp.to_csv(f'../data/price_histories/{ticker}_history.csv')
    print(f'{i+1} / {len(tickers)} completed', end='\r')
```

```
505 / 505 completed
```

Exploring Concepts and Prices

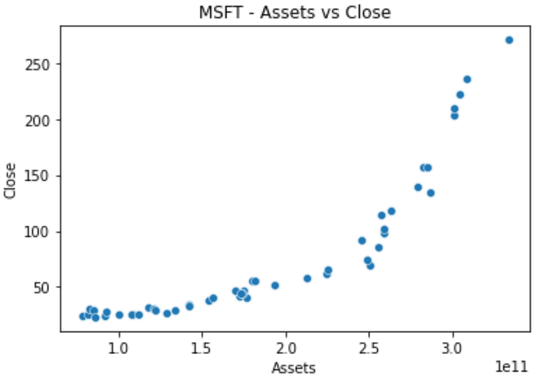
The idea of regressing certain pieces of fundamental data for a given company against its share price is at the core of this project. The function below allows for plotting a quick scatterplot for a given company's share price and any one of its concepts listed in the output of calling `get_company_concepts()` with its ticker as an argument.

In [13]:

```
def plot_scatter(ticker, concept, adjusted_price=False):
    '''
    Description
    -----------
    Plots a scatter plot for a given ticker's share price and
    a given concept.

    Parameters
    -----------
    ticker : str
        A company's ticker symbol for their publicly traded stock.

    concept : str
        The name of the concept. Must be available for the
        specified ticker. See `get_company_concepts()`.

    adjusted_price : bool, default=False
        Whether to use the adjusted closing price (adjusted for
        both dividends and splits) or the nominal closing price
        (adjusted for splits only).

    Example
    -----------
    >>> plot_scatter('MSFT', 'Assets')
    <matplotlib.axes._subplots.AxesSubplot>
    '''
    df_concept = get_concept_values(ticker, concept, as_df=True)
    df_price = pd.read_csv(f'../data/price_histories/{ticker}_history.csv',
                           index_col='Date',
                           parse_dates=['Date'])

    target_col = 'Close' if not adjusted_price else 'Adj Close'
    df_price = df_price[[target_col]]

    df_merged = pd.merge_asof(left=df_concept,
                              right=df_price,
                              left_index=True,
                              right_index=True)

    sns.scatterplot(x=concept,
                    y=target_col,
                    data=df_merged)
    plt.title(f'{ticker} - {concept} vs {target_col}')
```

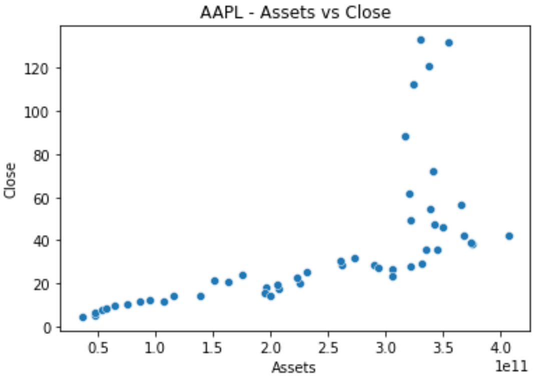
In [14]:

```
plot_scatter('MSFT', 'Assets')
```



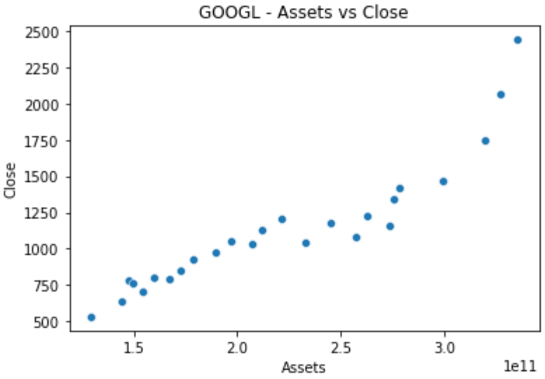
In [136]:

```
plot_scatter('AAPL', 'Assets')
```



In [137]:

```
plot_scatter('GOOGL', 'Assets')
```



Common Concepts

The number of concepts output by the `get_company_concepts()` function is fairly large for any particular company. Generating a list of only those concepts shared by all companies in the S&P 500 may significantly reduce the number and result in better comparisons, reduced computation times, and simpler models.

In [19]:

```
all_concepts = []
for i, ticker in enumerate(tickers):
    print(f'{i+1} / {len(tickers)}', end='\r')
    try:
        all_concepts.append(list(get_company_concepts(ticker)))
    except FileNotFoundError as e:
        print(f'Issue with ticker #{i+1} ({ticker}): {e}')
        pass
```

```
Issue with ticker #193 (FRC): [Errno 2] No such file or directory: '../data/sec_bulk_data/CIK0001132979.json'
505 / 505
```

In [14]:

```
set.intersection(*[set(c_list) for c_list in all_concepts])
```

Out[14]:

```
{'Assets', 'LiabilitiesAndStockholdersEquity'}
```

Unfortunately, the output above indicates that there are only two concepts that are shared across all of the companies. Since the project will require more than just that, it's necessary to find an appropriate threshold for the number of concepts vs number of companies sharing those concepts.

Number of Unique Concepts

The number below represents the number of unique concepts available across all companies in the S&P 500 (excluding FRC which ran into an error above).

In [20]:

```
c = Counter(item for sublist in all_concepts for item in sublist)
len(c)
```

Out[20]:

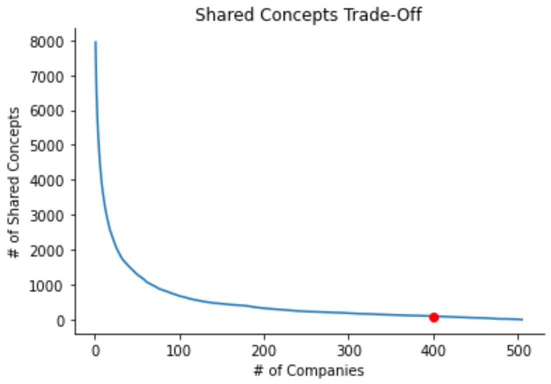
```
7941
```

Plotting the Trade-Off

Visualizing the trade-off between the desired number of shared concepts and the number of companies can assist in determining an optimal threshold value.

In [21]:

```
x = [i for i in range(1, len(tickers)+1)]
y = [len([item for item in c.items() if item[1] >= i]) for i in x]

sns.lineplot(x, y)
plt.plot(400, y[400], marker='o', color='red')
plt.ylabel('# of Shared Concepts')
plt.xlabel('# of Companies')
plt.title('Shared Concepts Trade-Off')
sns.despine();
```



The interpretation of this chart can be a bit confusing. The red dot above is placed at (400, 97) and implies that at a threshold of 400 companies, only 97 concepts remain common to all of them. The relationship is naturally inverse since few concepts apply to every single company. However, the minimum number of shared concepts being 2 is likely an artifact of missing / incomplete data from the SEC's bulk data download as there should certainly be more concepts that are shared by all (Assets, Liabilities, Equity, Revenue, Net Income, etc.).

Fetching Shared Concepts

With the relationship between the number of shared concepts for a given number of companies understood, it would be useful to have a function that returns a list of the shared concepts. This will allow for selecting a desired number of companies with comparable concepts and immediately seeing what those concepts are.

In [24]:

```python
def get_shared_concepts(counter, num_companies: int):
    '''
    Description
    -----------
    Given a specified number of companies, returns a list of the
    concepts shared by that number of companies. Exhibits an
    inverse relationship - the higher the number of companies, the
    lower the number of concepts shared among that many.

    Parameters
    -----------
    counter : collections.Counter object
        The Counter object storing the information on the number
        of companies with a specific concept.

    num_companies : int
        The number of companies.

    Example
    -----------
    >>> get_shared_concepts(num_companies=500)
    ['Assets',
     'EarningsPerShareBasic',
     'EarningsPerShareDiluted',
     'IncomeTaxExpenseBenefit',
     'LiabilitiesAndStockholdersEquity',
     'NetCashProvidedByUsedInFinancingActivities',
     'NetCashProvidedByUsedInInvestingActivities',
     'NetCashProvidedByUsedInOperatingActivities']
    '''
    return [item[0] for item in counter.items() if item[1] >= num_companies]

shared_concepts = get_shared_concepts(c, 470)
shared_concepts
```

Out[24]:

```
['AccumulatedOtherComprehensiveIncomeLossNetOfTax',
 'Assets',
 'CashAndCashEquivalentsAtCarryingValue',
 'CashAndCashEquivalentsPeriodIncreaseDecrease',
 'CommonStockSharesAuthorized',
 'ComprehensiveIncomeNetOfTax',
 'DeferredIncomeTaxExpenseBenefit',
 'EarningsPerShareBasic',
 'EarningsPerShareDiluted',
 'Goodwill',
 'IncomeTaxExpenseBenefit',
 'LesseeOperatingLeaseLiabilityPaymentsDue',
 'LesseeOperatingLeaseLiabilityUndiscountedExcessAmount',
 'LiabilitiesAndStockholdersEquity',
 'NetCashProvidedByUsedInFinancingActivities',
 'NetCashProvidedByUsedInInvestingActivities',
 'NetCashProvidedByUsedInOperatingActivities',
 'NetIncomeLoss',
 'OperatingLeaseLiability',
 'OperatingLeaseRightOfUseAsset',
 'OperatingLeasesFutureMinimumPaymentsDueCurrent',
 'OperatingLeasesFutureMinimumPaymentsDueInFiveYears',
 'OperatingLeasesFutureMinimumPaymentsDueInFourYears',
 'OperatingLeasesFutureMinimumPaymentsDueInThreeYears',
 'OperatingLeasesFutureMinimumPaymentsDueInTwoYears',
 'OperatingLeasesFutureMinimumPaymentsDueThereafter',
 'PropertyPlantAndEquipmentNet',
 'RetainedEarningsAccumulatedDeficit',
 'StockholdersEquity',
 'UnrecognizedTaxBenefits',
 'WeightedAverageNumberOfDilutedSharesOutstanding',
 'WeightedAverageNumberOfSharesOutstandingBasic']
```

Exporting Selected Data for Each Company

Testing with MSFT

In [69]:

```
df_msft = pd.DataFrame()
for concept in shared_concepts:
    if concept in get_company_concepts('MSFT'):
        try:
            values = get_concept_values('MSFT', concept, as_df=True)[concept]
            df_msft[concept] = values
        except KeyError as e:
            print(f"Concept '{concept}' has a non-USD unit")
```

```
Concept 'CommonStockSharesAuthorized' has a non-USD unit
Concept 'WeightedAverageNumberOfDilutedSharesOutstanding' has a non-USD unit
Concept 'WeightedAverageNumberOfSharesOutstandingBasic' has a non-USD unit
```

In [70]:

```
df_msft.head()
```

Out[70]:

	AccumulatedOtherComprehensiveIncomeLossNetOfTax	Assets	CashAndCashEquivalentsAtCarryingValue	CashAndCashEquivalentsPeriodIncreaseDecrease
2008-06-30	1140000000	NaN	10339000000	NaN
2009-06-30	969000000	7.788800e+10	6076000000	NaN
2009-09-30	1334000000	8.161200e+10	8823000000	2.747000e+09
2009-12-31	1322000000	8.209600e+10	9422000000	5.990000e+08
2010-03-31	1453000000	8.491000e+10	8155000000	-1.267000e+09

5 rows × 23 columns

In [71]:

```
df_msft_price = pd.read_csv(f'../data/price_histories/MSFT_history.csv',
                            index_col='Date',
                            parse_dates=['Date'])
df_msft_price.head()
```

Out[71]:

	Open	High	Low	Close	Adj Close	Volume
Date						
1986-03-13	0.088542	0.101563	0.088542	0.097222	0.061608	1.031789e+09
1986-03-14	0.097222	0.102431	0.097222	0.100694	0.063809	3.081600e+08
1986-03-17	0.100694	0.103299	0.100694	0.102431	0.064909	1.331712e+08
1986-03-18	0.102431	0.103299	0.098958	0.099826	0.063258	6.776640e+07
1986-03-19	0.099826	0.100694	0.097222	0.098090	0.062158	4.789440e+07

In [72]:

```
df_msft = pd.merge_asof(left=df_msft,
                        right=df_msft_price.Close,
                        left_index=True,
                        right_index=True)
df_msft.head()
```

Out[72]:

	AccumulatedOtherComprehensiveIncomeLossNetOfTax	Assets	CashAndCashEquivalentsAtCarryingValue	CashAndCashEquivalentsPeriodIncreaseDecrease
2008-06-30	1140000000	NaN	10339000000	NaN
2009-06-30	969000000	7.788800e+10	6076000000	NaN
2009-09-30	1334000000	8.161200e+10	8823000000	2.747000e+09
2009-12-31	1322000000	8.209600e+10	9422000000	5.990000e+08
2010-03-31	1453000000	8.491000e+10	8155000000	-1.267000e+09

5 rows × 24 columns

In [74]:

```
df_msft.rename(columns={'Close': 'SharePrice'}, inplace=True)
```

In [75]:

```
df_msft.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 50 entries, 2008-06-30 to 2021-06-30
Data columns (total 24 columns):
 #   Column                                            Non-Null Count  Dtype  
---  ------                                            --------------  -----  
 0   AccumulatedOtherComprehensiveIncomeLossNetOfTax   50 non-null     int64  
 1   Assets                                            49 non-null     float64
 2   CashAndCashEquivalentsAtCarryingValue             50 non-null     int64  
 3   CashAndCashEquivalentsPeriodIncreaseDecrease      30 non-null     float64
 4   ComprehensiveIncomeNetOfTax                       30 non-null     float64
 5   DeferredIncomeTaxExpenseBenefit                   36 non-null     float64
 6   EarningsPerShareBasic                             48 non-null     float64
 7   EarningsPerShareDiluted                           48 non-null     float64
 8   Goodwill                                          50 non-null     int64  
 9   IncomeTaxExpenseBenefit                           36 non-null     float64
 10  LesseeOperatingLeaseLiabilityPaymentsDue          16 non-null     float64
 11  LesseeOperatingLeaseLiabilityUndiscountedExcessAmount  16 non-null     float64
 12  LiabilitiesAndStockholdersEquity                  49 non-null     float64
 13  NetCashProvidedByUsedInFinancingActivities        26 non-null     float64
 14  NetCashProvidedByUsedInInvestingActivities        26 non-null     float64
 15  NetCashProvidedByUsedInOperatingActivities        26 non-null     float64
 16  NetIncomeLoss                                     49 non-null     float64
 17  OperatingLeaseLiability                           18 non-null     float64
 18  OperatingLeaseRightOfUseAsset                     18 non-null     float64
 19  PropertyPlantAndEquipmentNet                      49 non-null     float64
 20  RetainedEarningsAccumulatedDeficit                43 non-null     float64
 21  StockholdersEquity                                50 non-null     int64  
 22  UnrecognizedTaxBenefits                           14 non-null     float64
 23  SharePrice                                        50 non-null     float64
dtypes: float64(20), int64(4)
memory usage: 9.8 KB
```