



PANDAS FOUNDATIONS

# **pandas Foundations**

# What is pandas?

- Python library for data analysis
- High-performance containers for data analysis
- Data structures with a lot of functionality
  - Meaningful labels
  - Time series functionality
  - Handling missing data
  - Relational operations



# What you will learn

- How to work with pandas
  - Data import & export in various formats
- Exploratory Data Analysis using pandas
  - Statistical & graphical methods
- Using pandas to model *time series*
  - Time indexes, resampling



PANDAS FOUNDATIONS

**See you in  
the course!**



PANDAS FOUNDATIONS

# **Review of pandas DataFrames**



# pandas DataFrames

- Example: DataFrame of Apple Stock data

Date	Open	High	Low	Close	Volume	Adj Close
2014-09-16	99.80	101.26	98.89	100.86	66818200	100.86
2014-09-15	102.81	103.05	101.44	101.63	61216500	101.63
2014-09-12	101.21	102.19	101.08	101.66	62626100	101.66
...	...	...	...	...	...	...



# Indexes and columns

```
In [1]: import pandas as pd
```

```
In [2]: type(AAPL)
```

```
Out[2]: pandas.core.frame.DataFrame
```

```
In [3]: AAPL.shape
```

```
Out[3]: (8514, 6)
```

```
In [4]: AAPL.columns
```

```
Out[4]:
```

```
Index(['Open', 'High', 'Low', 'Close', 'Volume', 'Adj Close'],  
      dtype='object')
```

```
In [5]: type(AAPL.columns)
```

```
Out[5]: pandas.indexes.base.Index
```



# Indexes and columns

```
In [6]: AAPL.index
```

```
Out[6]:
```

```
DatetimeIndex(['2014-09-16', '2014-09-15', '2014-09-12',  
              '2014-09-11', '2014-09-10', '2014-09-09',  
              '2014-09-08', '2014-09-05', '2014-09-04',  
              '2014-09-03',  
              ...  
              '1980-12-26', '1980-12-24', '1980-12-23',  
              '1980-12-22', '1980-12-19', '1980-12-18',  
              '1980-12-17', '1980-12-16', '1980-12-15',  
              '1980-12-12'],  
              dtype='datetime64[ns]', name='Date', length=8514,  
              freq=None)
```

```
In [7]: type(AAPL.index)
```

```
Out[7]: pandas.tseries.index.DatetimeIndex
```





# Slicing

```
In [8]: AAPL.iloc[:5,:]
```

```
Out[8]:
```

	Open	High	Low	Close	Volume	Adj Close
Date						
2014-09-16	99.80	101.26	98.89	100.86	66818200	100.86
2014-09-15	102.81	103.05	101.44	101.63	61216500	101.63
2014-09-12	101.21	102.19	101.08	101.66	62626100	101.66
2014-09-11	100.41	101.44	99.62	101.43	62353100	101.43
2014-09-10	98.01	101.11	97.76	101.00	100741900	101.00

```
In [9]: AAPL.iloc[-5:,:]
```

```
Out[9]:
```

	Open	High	Low	Close	Volume	Adj Close
Date						
1980-12-18	26.63	26.75	26.63	26.63	18362400	0.41
1980-12-17	25.87	26.00	25.87	25.87	21610400	0.40
1980-12-16	25.37	25.37	25.25	25.25	26432000	0.39
1980-12-15	27.38	27.38	27.25	27.25	43971200	0.42
1980-12-12	28.75	28.87	28.75	28.75	117258400	0.45



# head()

```
In [10]: AAPL.head(5)
```

```
Out[10]:
```

	Open	High	Low	Close	Volume	Adj Close
Date						
2014-09-16	99.80	101.26	98.89	100.86	66818200	100.86
2014-09-15	102.81	103.05	101.44	101.63	61216500	101.63
2014-09-12	101.21	102.19	101.08	101.66	62626100	101.66
2014-09-11	100.41	101.44	99.62	101.43	62353100	101.43
2014-09-10	98.01	101.11	97.76	101.00	100741900	101.00

```
In [11]: AAPL.head(2)
```

```
Out[11]:
```

	Open	High	Low	Close	Volume	Adj Close
Date						
2014-09-16	99.80	101.26	98.89	100.86	66818200	100.86
2014-09-15	102.81	103.05	101.44	101.63	61216500	101.63



# tail()

```
In [12]: AAPL.tail()
```

```
Out[12]:
```

	Open	High	Low	Close	Volume	Adj Close
Date						
1980-12-18	26.63	26.75	26.63	26.63	18362400	0.41
1980-12-17	25.87	26.00	25.87	25.87	21610400	0.40
1980-12-16	25.37	25.37	25.25	25.25	26432000	0.39
1980-12-15	27.38	27.38	27.25	27.25	43971200	0.42
1980-12-12	28.75	28.87	28.75	28.75	117258400	0.45

```
In [13]: AAPL.tail(3)
```

```
Out[13]:
```

	Open	High	Low	Close	Volume	Adj Close
Date						
1980-12-16	25.37	25.37	25.25	25.25	26432000	0.39
1980-12-15	27.38	27.38	27.25	27.25	43971200	0.42
1980-12-12	28.75	28.87	28.75	28.75	117258400	0.45



# info()

```
In [14]: AAPL.info()
Out[14]:
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 8514 entries, 2014-09-16 to 1980-12-12
Data columns (total 6 columns):
Open           8514 non-null float64
High           8514 non-null float64
Low            8514 non-null float64
Close          8514 non-null float64
Volume         8514 non-null int64
Adj Close      8514 non-null float64
dtypes: float64(5), int64(1)
memory usage: 465.6 KB
```



# Broadcasting

```
In [15]: import numpy as np
```

```
In [16]: AAPL.iloc[:, 3, -1] = np.nan
```

← Assigning scalar value to column slice *broadcasts* value to each row.

```
In [17]: AAPL.head(6)
```

```
Out[17]:
```

	Open	High	Low	Close	Volume	Adj Close
Date						
2014-09-16	99.80	101.26	98.89	100.86	66818200	NaN
2014-09-15	102.81	103.05	101.44	101.63	61216500	101.63
2014-09-12	101.21	102.19	101.08	101.66	62626100	101.66
2014-09-11	100.41	101.44	99.62	101.43	62353100	NaN
2014-09-10	98.01	101.11	97.76	101.00	100741900	101.00
2014-09-09	99.08	103.08	96.14	97.99	189560600	97.99
2014-09-08	99.30	99.31	98.05	98.36	46277800	NaN



# Broadcasting

```
In [18]: AAPL.info()
Out[18]:
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 8514 entries, 2014-09-16 to 1980-12-12
Data columns (total 6 columns):
Open                8514 non-null float64
High                8514 non-null float64
Low                 8514 non-null float64
Close               8514 non-null float64
Volume              8514 non-null int64
Adj Close           5676 non-null float64
dtypes: float64(5), int64(1)
memory usage: 465.6 KB
```



# Series

```
In [19]: low = AAPL['Low']
```

```
In [20]: type(low)
```

```
Out[20]: pandas.core.series.Series
```

```
In [21]: low.head()
```

```
Out[21]:
```

Date

2014-09-16	98.89
------------	-------

2014-09-15	101.44
------------	--------

2014-09-12	101.08
------------	--------

2014-09-11	99.62
------------	-------

2014-09-10	97.76
------------	-------

Name: Low, dtype: float64

```
In [22]: lows = low.values
```

```
In [23]: type(lows)
```

```
Out[23]: numpy.ndarray
```



PANDAS FOUNDATIONS

**Let's practice!**





PANDAS FOUNDATIONS

# **Building DataFrames from scratch**



# DataFrames from CSV files

```
In [1]: import pandas as pd
```

```
In [2]: users = pd.read_csv('datasets/users.csv', index_col=0)
```

```
In [3]: print(users)
```

```
Out[3]:
```

	weekday	city	visitors	signups
0	Sun	Austin	139	7
1	Sun	Dallas	237	12
2	Mon	Austin	326	3
3	Mon	Dallas	456	5



# DataFrames from dict (1)

```
In [1]: import pandas as pd
```

```
In [2]: data = {'weekday': ['Sun', 'Sun', 'Mon', 'Mon'],  
...:           'city': ['Austin', 'Dallas', 'Austin', 'Dallas'],  
...:           'visitors': [139, 237, 326, 456],  
...:           'signups': [7, 12, 3, 5]}
```

```
In [3]: users = pd.DataFrame(data)
```

```
In [4]: print(users)
```

```
Out[4]:
```

	weekday	city	visitors	signups
0	Sun	Austin	139	7
1	Sun	Dallas	237	12
2	Mon	Austin	326	3
3	Mon	Dallas	456	5



# DataFrames from dict (2)

```
In [1]: import pandas as pd
```

```
In [2]: cities = ['Austin', 'Dallas', 'Austin', 'Dallas']
```

```
In [3]: signups = [7, 12, 3, 5]
```

```
In [4]: visitors = [139, 237, 326, 456]
```

```
In [5]: weekdays = ['Sun', 'Sun', 'Mon', 'Mon']
```

```
In [6]: list_labels = ['city', 'signups', 'visitors', 'weekday']
```

```
In [7]: list_cols = [cities, signups, visitors, weekdays]
```

```
In [8]: zipped = list(zip(list_labels, list_cols))
```



# DataFrames from dict (3)

```
In [9]: print(zipped)
```

```
Out[9]:
```

```
[('city', ['Austin', 'Dallas', 'Austin', 'Dallas']), ('signups',  
[7, 12, 3, 5]), ('visitors', [139, 237, 326, 456]), ('weekday',  
['Sun', 'Sun', 'Mon', 'Mon'])]
```

```
In [10]: data = dict(zipped)
```

```
In [11]: users = pd.DataFrame(data)
```

```
In [12]: print(users)
```

```
Out[12]:
```

	weekday	city	visitors	signups
0	Sun	Austin	139	7
1	Sun	Dallas	237	12
2	Mon	Austin	326	3
3	Mon	Dallas	456	5



# Broadcasting

```
In [13]: users['fees'] = 0 # Broadcasts to entire column
```

```
In [14]: print(users)
```

```
Out[14]:
```

	city	signups	visitors	weekday	fees
0	Austin	7	139	Sun	0
1	Dallas	12	237	Sun	0
2	Austin	3	326	Mon	0
3	Dallas	5	456	Mon	0



# Broadcasting with a dict

```
In [1]: import pandas as pd
```

```
In [2]: heights = [ 59.0, 65.2, 62.9, 65.4, 63.7, 65.7, 64.1 ]
```

```
In [3]: data = {'height': heights, 'sex': 'M'}
```

```
In [4]: results = pd.DataFrame(data)
```

```
In [5]: print(results)
```

```
Out[5]:
```

	height	sex
0	59.0	M
1	65.2	M
2	62.9	M
3	65.4	M
4	63.7	M
5	65.7	M
6	64.1	M



# Index and columns

```
In [6]: results.columns = ['height (in)', 'sex']
```

```
In [7]: results.index = ['A', 'B', 'C', 'D', 'E', 'F', 'G']
```

```
In [8]: print(results)
```

```
Out[8]:
```

	height (in)	sex
A	59.0	M
B	65.2	M
C	62.9	M
D	65.4	M
E	63.7	M
F	65.7	M
G	64.1	M





PANDAS FOUNDATIONS

**Let's practice!**



PANDAS FOUNDATIONS

# Importing & exporting data



# Original CSV file

- Dataset: Sunspot observations collected from SILSO

```
1818,01,01,1818.004, -1,1
1818,01,02,1818.007, -1,1
1818,01,03,1818.010, -1,1
1818,01,04,1818.012, -1,1
1818,01,05,1818.015, -1,1
1818,01,06,1818.018, -1,1
...
```



# Datasets from CSV files

```
In [1]: import pandas as pd
```

```
In [2]: filepath = 'ISSN_D_tot.csv'
```

```
In [3]: sunspots = pd.read_csv(filepath)
```

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

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 71921 entries, 0 to 71920
```

```
Data columns (total 6 columns):
```

1818	71921 non-null	int64
01	71921 non-null	int64
01.1	71921 non-null	int64
1818.004	71921 non-null	float64
-1	71921 non-null	int64
1	71921 non-null	int64

```
dtypes: float64(1), int64(5)
```

```
memory usage: 3.3 MB
```



# Datasets from CSV files

```
In [5]: sunspots.iloc[10:20, :]
```

```
Out[5]:
```

	1818	01	01.1	1818.004	-1	1
10	1818	1	12	1818.034	-1	1
11	1818	1	13	1818.037	22	1
12	1818	1	14	1818.040	-1	1
13	1818	1	15	1818.042	-1	1
14	1818	1	16	1818.045	-1	1
15	1818	1	17	1818.048	46	1
16	1818	1	18	1818.051	59	1
17	1818	1	19	1818.053	63	1
18	1818	1	20	1818.056	-1	1
19	1818	1	21	1818.059	-1	1

# Problems

- CSV file has no column headers
  - Columns 0-2: Gregorian date (year, month, day)
  - Column 3: Date as fraction as year
  - Column 4: Daily total sunspot number
  - Column 5: Definitive/provisional indicator (1 or 0)
- Missing values in column 4: indicated by -1
- Dates representation inconvenient



# Using header keyword

```
In [6]: sunspots = pd.read_csv(filepath, header=None)
```

```
In [7]: sunspots.iloc[10:20, :]
```

```
Out[7]:
```

	0	1	2	3	4	5
10	1818	1	11	1818.031	-1	1
11	1818	1	12	1818.034	-1	1
12	1818	1	13	1818.037	22	1
13	1818	1	14	1818.040	-1	1
14	1818	1	15	1818.042	-1	1
15	1818	1	16	1818.045	-1	1
16	1818	1	17	1818.048	46	1
17	1818	1	18	1818.051	59	1
18	1818	1	19	1818.053	63	1
19	1818	1	20	1818.056	-1	1



# Using names keyword

```
In [8]: col_names = ['year', 'month', 'day', 'dec_date',  
....:                  'sunspots', 'definite']
```

```
In [9]: sunspots = pd.read_csv(filepath, header=None,  
....:                           names=col_names)
```

```
In [10]: sunspots.iloc[10:20, :]
```

```
Out[10]:
```

	year	month	day	dec_date	sunspots	definite
10	1818	1	11	1818.031	-1	1
11	1818	1	12	1818.034	-1	1
12	1818	1	13	1818.037	22	1
13	1818	1	14	1818.040	-1	1
14	1818	1	15	1818.042	-1	1
15	1818	1	16	1818.045	-1	1
16	1818	1	17	1818.048	46	1
17	1818	1	18	1818.051	59	1
18	1818	1	19	1818.053	63	1
19	1818	1	20	1818.056	-1	1





# Using na\_values keyword (1)

```
In [11]: sunspots = pd.read_csv(filepath, header=None,  
    ....:                        names=col_names, na_values='-1')
```

```
In [12]: sunspots.iloc[10:20, :]
```

```
Out[12]:
```

	year	month	day	dec_date	sunspots	definite
10	1818	1	11	1818.031	-1	1
11	1818	1	12	1818.034	-1	1
12	1818	1	13	1818.037	22	1
13	1818	1	14	1818.040	-1	1
14	1818	1	15	1818.042	-1	1
15	1818	1	16	1818.045	-1	1
16	1818	1	17	1818.048	46	1
17	1818	1	18	1818.051	59	1
18	1818	1	19	1818.053	63	1
19	1818	1	20	1818.056	-1	1



# Using na\_values keyword (2)

```
In [13]: sunspots = pd.read_csv(filepath, header=None,  
    ....:                        names=col_names, na_values=' -1')
```

```
In [14]: sunspots.iloc[10:20, :]
```

```
Out[14]:
```

	year	month	day	dec_date	sunspots	definite
10	1818	1	11	1818.031	NaN	1
11	1818	1	12	1818.034	NaN	1
12	1818	1	13	1818.037	22.0	1
13	1818	1	14	1818.040	NaN	1
14	1818	1	15	1818.042	NaN	1
15	1818	1	16	1818.045	NaN	1
16	1818	1	17	1818.048	46.0	1
17	1818	1	18	1818.051	59.0	1
18	1818	1	19	1818.053	63.0	1
19	1818	1	20	1818.056	NaN	1



# Using na\_values keyword (3)

```
In [15]: sunspots = pd.read_csv(filepath, header=None,  
    ...: names=col_names, na_values={'sunspots': ['-1']})
```

```
In [16]: sunspots.iloc[10:20, :]
```

```
Out[16]:
```

	year	month	day	dec_date	sunspots	definite
10	1818	1	11	1818.031	NaN	1
11	1818	1	12	1818.034	NaN	1
12	1818	1	13	1818.037	22.0	1
13	1818	1	14	1818.040	NaN	1
14	1818	1	15	1818.042	NaN	1
15	1818	1	16	1818.045	NaN	1
16	1818	1	17	1818.048	46.0	1
17	1818	1	18	1818.051	59.0	1
18	1818	1	19	1818.053	63.0	1
19	1818	1	20	1818.056	NaN	1



# Using parse\_dates keyword

```
In [17]: sunspots = pd.read_csv(filepath, header=None,  
...: names=col_names, na_values={'sunspots': ['-1']},  
...: parse_dates=[[0, 1, 2]])
```

```
In [18]: sunspots.iloc[10:20, :]
```

```
Out[18]:
```

	year_month_day	dec_date	sunspots	definite
10	1818-01-11	1818.031	NaN	1
11	1818-01-12	1818.034	NaN	1
12	1818-01-13	1818.037	22.0	1
13	1818-01-14	1818.040	NaN	1
14	1818-01-15	1818.042	NaN	1
15	1818-01-16	1818.045	NaN	1
16	1818-01-17	1818.048	46.0	1
17	1818-01-18	1818.051	59.0	1
18	1818-01-19	1818.053	63.0	1
19	1818-01-20	1818.056	NaN	1



# Inspecting DataFrame

```
In [19]: sunspots.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 71922 entries, 0 to 71921
Data columns (total 4 columns):
year_month_day      71922 non-null datetime64[ns]
dec_date            71922 non-null float64
sunspots            68675 non-null float64
definite            71922 non-null int64
dtypes: datetime64[ns](1), float64(2), int64(1)
memory usage: 2.2 MB
```



# Using dates as index

```
In [20]: sunspots.index = sunspots['year_month_day']
```

```
In [21]: sunspots.index.name = 'date'
```

```
In [22]: sunspots.info()
```

```
Out[22]:
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
DatetimeIndex: 71922 entries, 1818-01-01 to 2014-11-30
```

```
Data columns (total 4 columns):
```

```
year_month_day    71922 non-null datetime64[ns]
```

```
dec_date          71922 non-null float64
```

```
sunspots          68675 non-null float64
```

```
definite          71922 non-null int64
```

```
dtypes: datetime64[ns](1), float64(2), int64(1)
```

```
memory usage: 2.7 MB
```



# Trimming redundant columns

```
In [23]: cols = ['sunspots', 'definite']
```

```
In [24]: sunspots = sunspots[cols]
```

```
In [25]: sunspots.iloc[10:20, :]
```

```
Out[25]:
```

	sunspots	definite
date		
1818-01-11	NaN	1
1818-01-12	NaN	1
1818-01-13	22.0	1
1818-01-14	NaN	1
1818-01-15	NaN	1
1818-01-16	NaN	1
1818-01-17	46.0	1
1818-01-18	59.0	1
1818-01-19	63.0	1
1818-01-20	NaN	1



# Writing files

```
In [26]: out_csv = 'sunspots.csv'
```

```
In [27]: sunspots.to_csv(out_csv)
```

```
In [28]: out_tsv = 'sunspots.tsv'
```

```
In [29]: sunspots.to_csv(out_tsv, sep='\t')
```

```
In [30]: out_xlsx = 'sunspots.xlsx'
```

```
In [31]: sunspots.to_excel(out_xlsx)
```





PANDAS FOUNDATIONS

**Let's practice!**



PANDAS FOUNDATIONS

# Plotting with pandas



# AAPL stock data

```
In [1]: import pandas as pd
```

```
In [2]: import matplotlib.pyplot as plt
```

```
In [3]: aapl = pd.read_csv('aapl.csv', index_col='date',  
....:                      parse_dates=True)
```

```
In [4]: aapl.head(6)
```

```
Out[4]:
```

	adj_close	close	high	low	open	volume
date						
2000-03-01	31.68	130.31	132.06	118.50	118.56	38478000
2000-03-02	29.66	122.00	127.94	120.69	127.00	11136800
2000-03-03	31.12	128.00	128.23	120.00	124.87	11565200
2000-03-06	30.56	125.69	129.13	125.00	126.00	7520000
2000-03-07	29.87	122.87	127.44	121.12	126.44	9767600
2000-03-08	29.66	122.00	123.94	118.56	122.87	9690800



# Plotting arrays (matplotlib)

```
In [5]: close_arr = aapl['close'].values
```

```
In [6]: type(close_arr)
```

```
Out[6]: numpy.ndarray
```

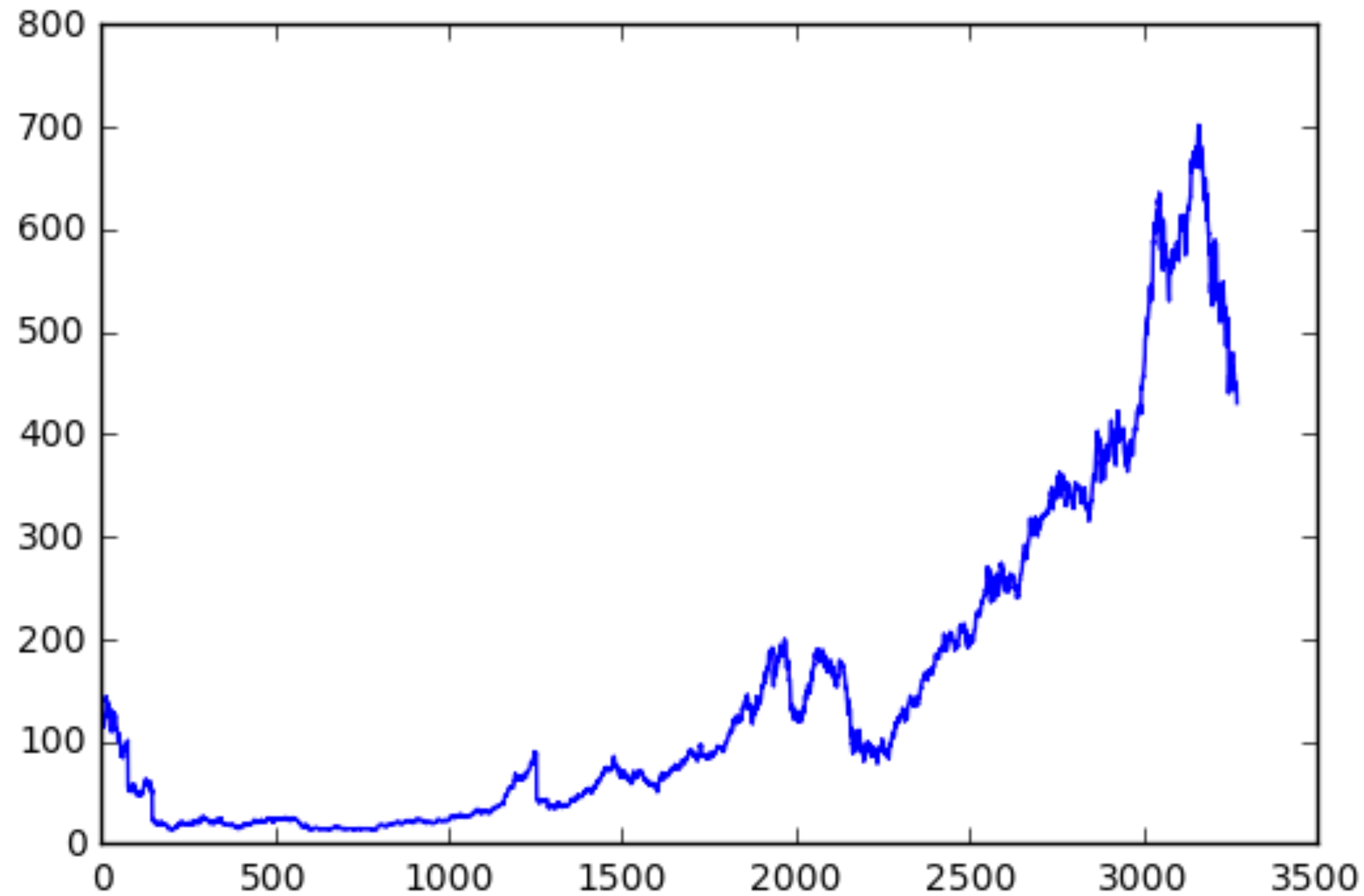
```
In [7]: plt.plot(close_arr)
```

```
Out[7]: [<matplotlib.lines.Line2D at 0x115550358>]
```

```
In [8]: plt.show()
```



# Plotting arrays (Matplotlib)





# Plotting Series (matplotlib)

```
In [9]: close_series = aapl['close']
```

```
In [10]: type(close_series)
```

```
Out[10]: pandas.core.series.Series
```

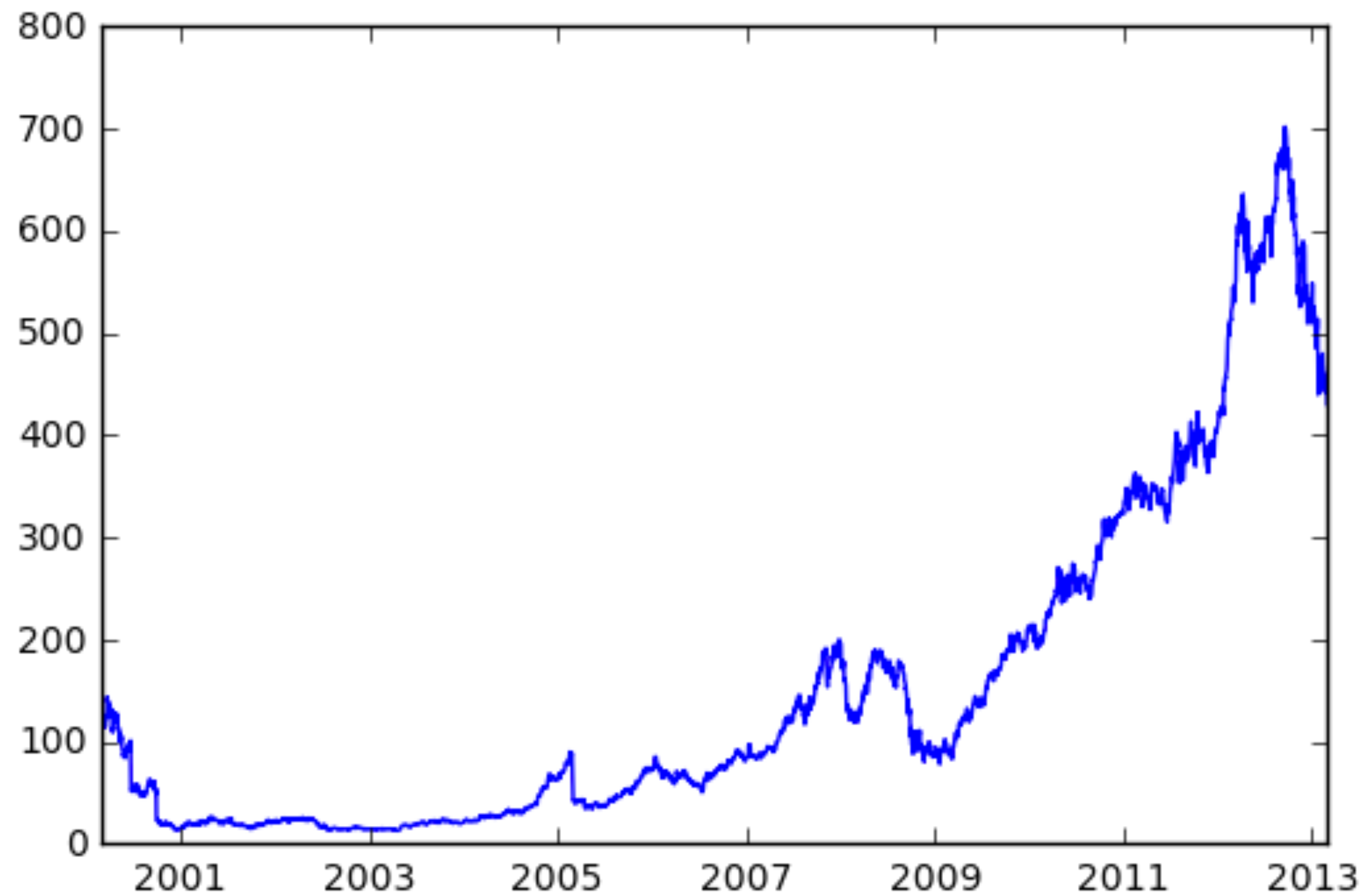
```
In [11]: plt.plot(close_series)
```

```
Out[11]: [<matplotlib.lines.Line2D at 0x11801cd30>]
```

```
In [12]: plt.show()
```



# Plotting Series (matplotlib)





# Plotting Series (pandas)

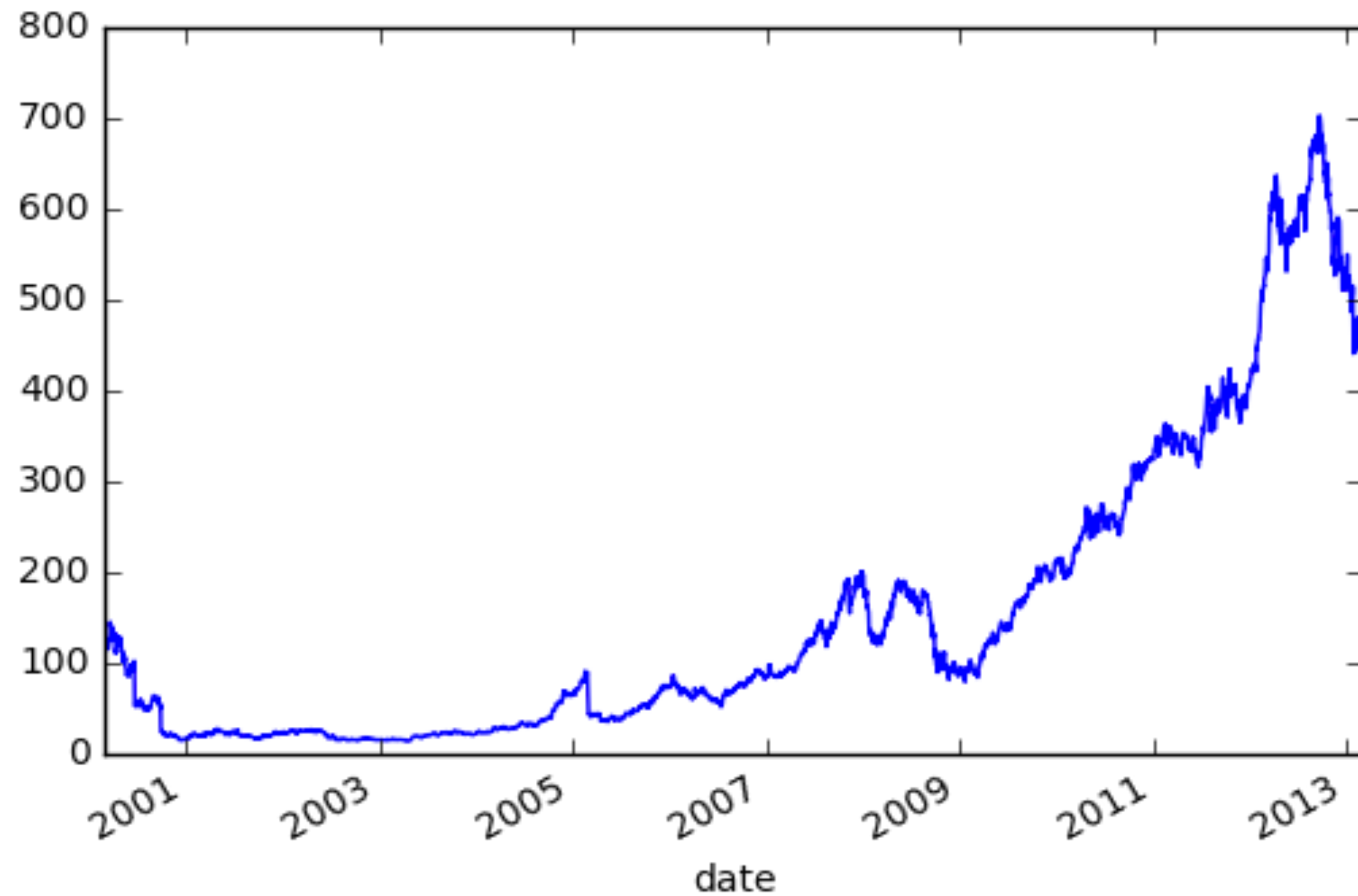
```
In [13]: close_series.plot() # plots Series directly
```

```
In [14]: plt.show()
```





# Plotting Series (pandas)

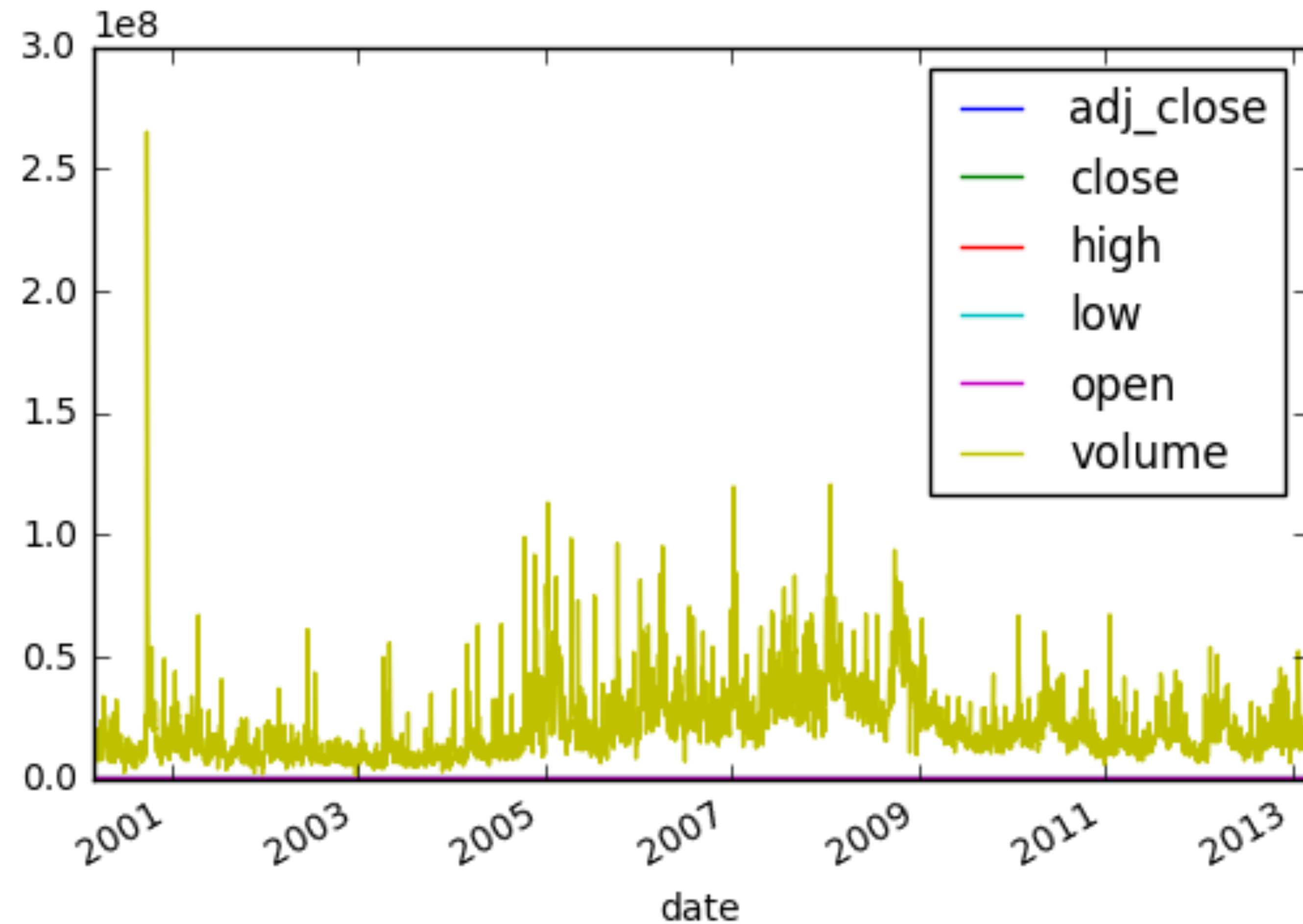


# Plotting DataFrames (pandas)

```
In [15]: aapl.plot() # plots all Series at once  
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x118039b38>  
  
In [16]: plt.show()
```



# Plotting DataFrames (pandas)



# Plotting DataFrames (matplotlib)

```
In [17]: plt.plot(aapl) # plots all columns at once
```

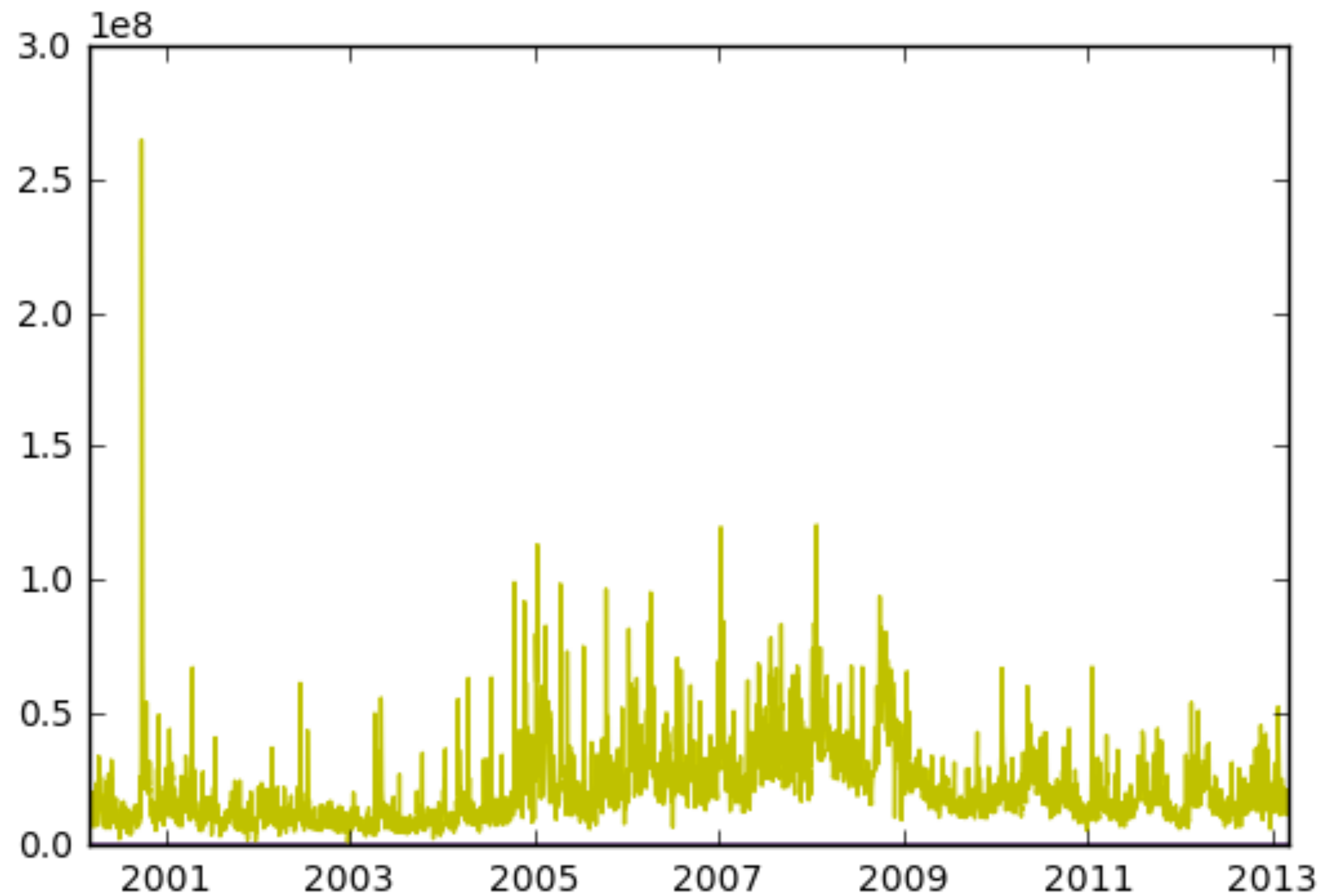
```
Out[17]:
```

```
<matplotlib.lines.Line2D at 0x1156290f0>,  
<matplotlib.lines.Line2D at 0x1156525f8>,  
<matplotlib.lines.Line2D at 0x1156527f0>,  
<matplotlib.lines.Line2D at 0x1156529e8>,  
<matplotlib.lines.Line2D at 0x115652be0>,  
<matplotlib.lines.Line2D at 0x115652dd8>
```

```
In [18]: plt.show()
```



# Plotting DataFrames (matplotlib)





# Fixing scales

```
In [19]: aapl.plot()
```

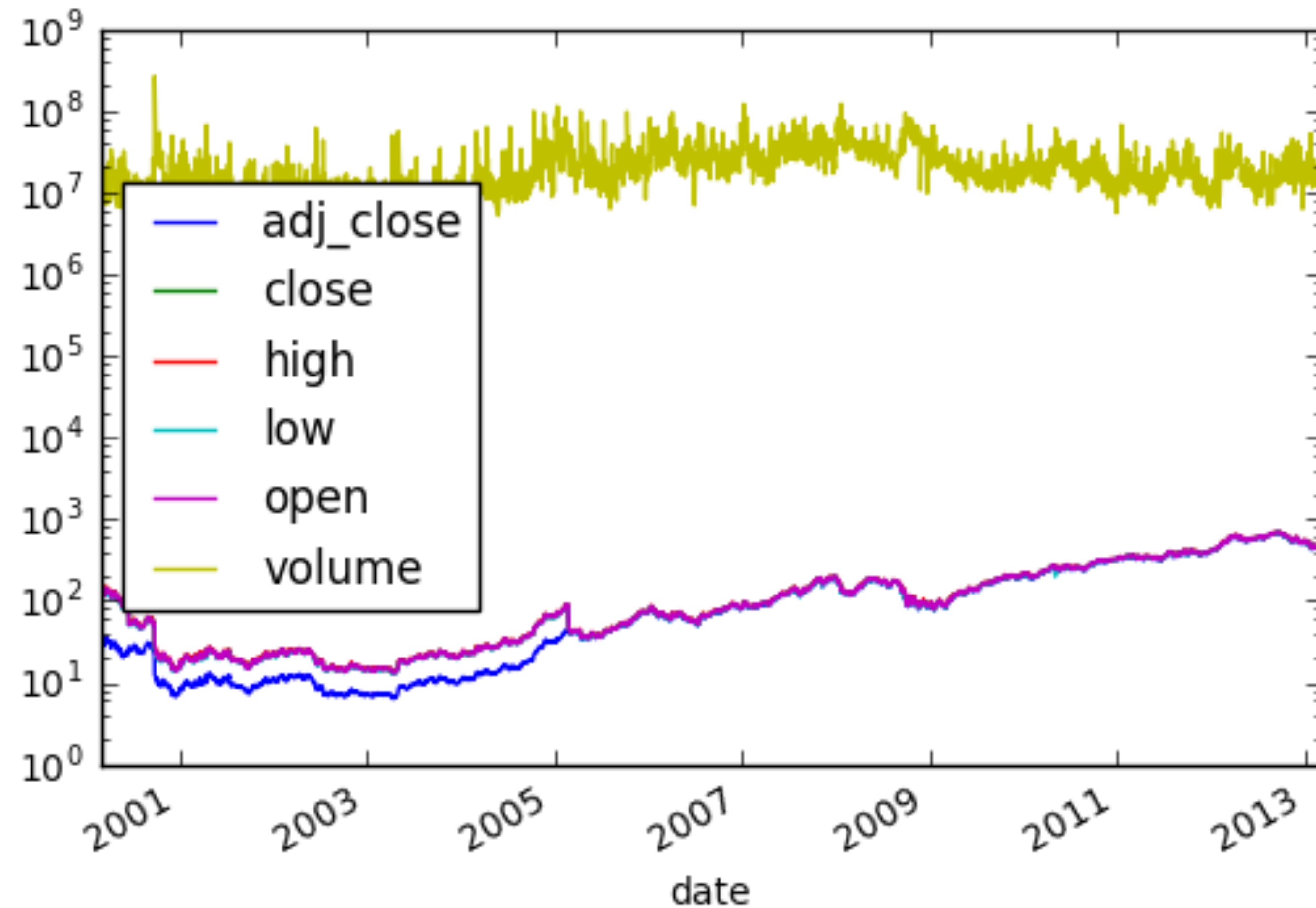
```
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x118afe048>
```

```
In [20]: plt.yscale('log') # logarithmic scale on vertical axis
```

```
In [21]: plt.show()
```



# Fixing scales





# Customizing plots

```
In [22]: aapl['open'].plot(color='b', style='.-', legend=True)
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x11a17db38>
```

```
In [23]: aapl['close'].plot(color='r', style='.', legend=True)
Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x11a17db38>
```

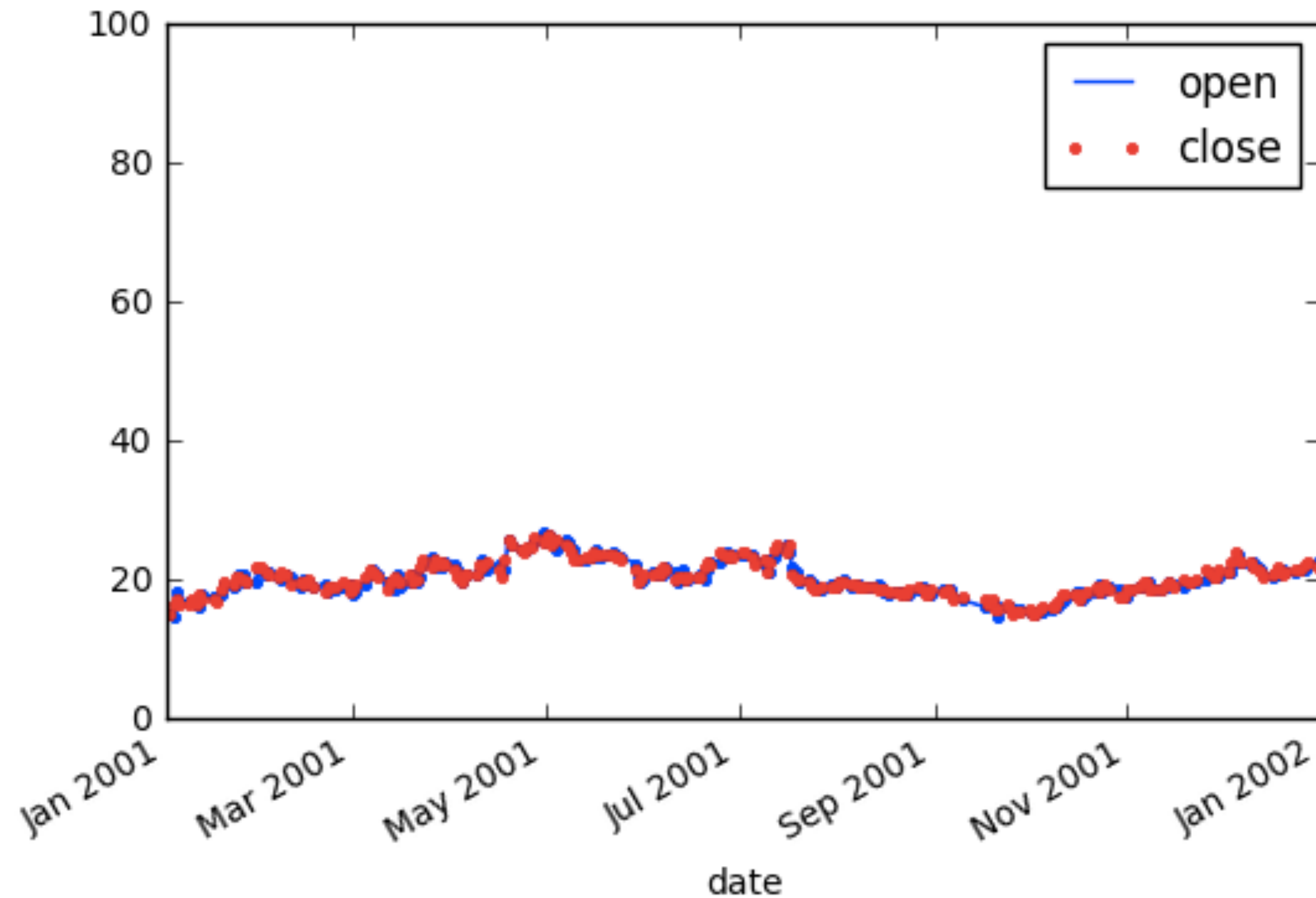
```
In [24]: plt.axis(('2001', '2002', 0, 100))
Out[24]: ('2001', '2002', 0, 100)
```

```
In [25]: plt.show()
```



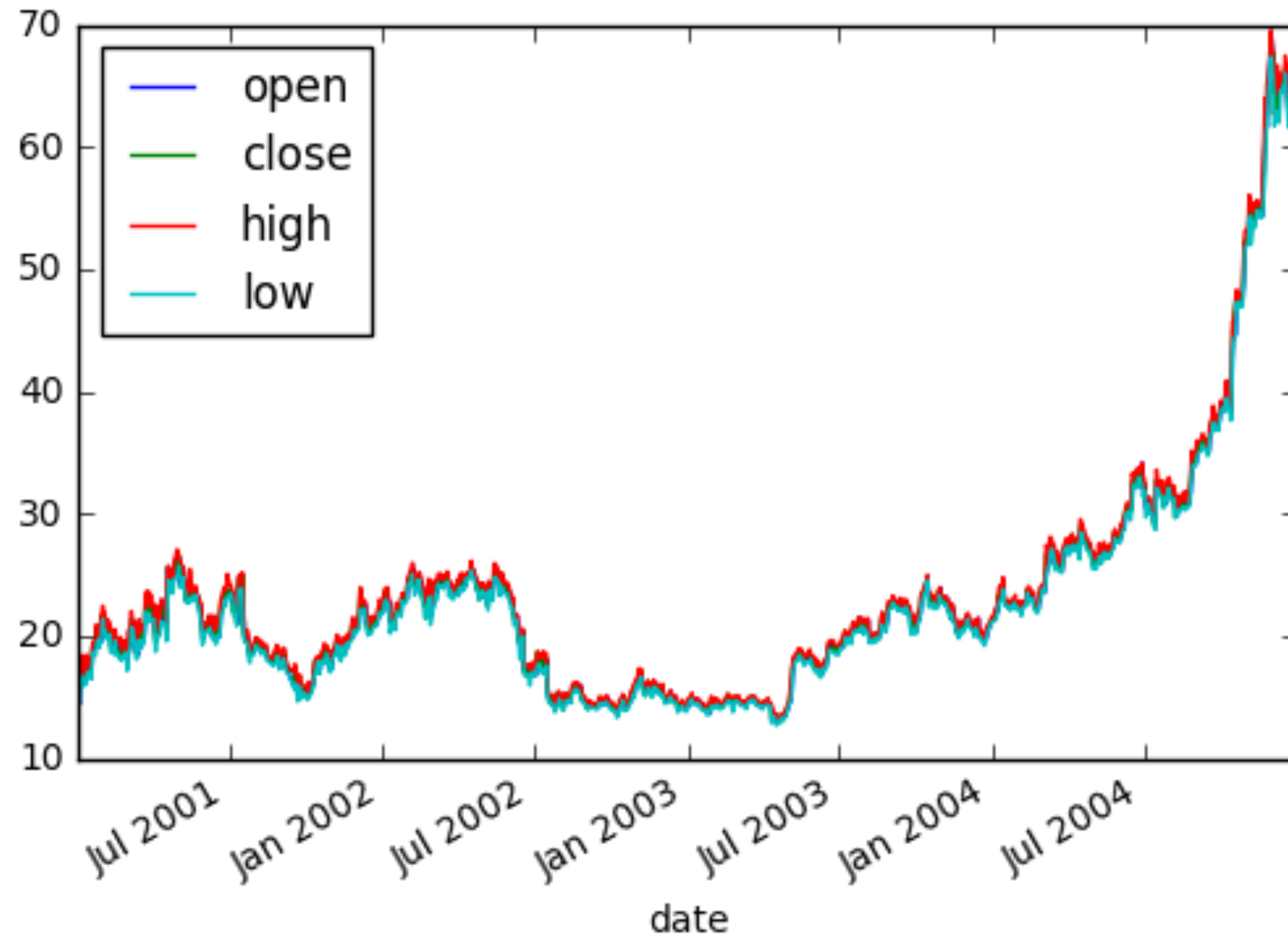


# Customizing plots





# Saving plots





# Saving plots

```
In [26]: aapl.loc['2001':'2004',['open', 'close', 'high',  
....:      'low']].plot()  
Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x11ab42978>  
  
In [27]: plt.savefig('aapl.png')  
  
In [28]: plt.savefig('aapl.jpg')  
  
In [29]: plt.savefig('aapl.pdf')  
  
In [30]: plt.show()
```



PANDAS FOUNDATIONS

**Let's practice!**



PANDAS FOUNDATIONS

# **Visual exploratory data analysis**



# The iris data set

- Famous data set in pattern recognition
- 150 observations, 4 features each
  - Sepal length
  - Sepal width
  - Petal length
  - Petal width
- 3 species: setosa, versicolor, virginica



# Data import

```
In [1]: import pandas as pd
```

```
In [2]: import matplotlib.pyplot as plt
```

```
In [3]: iris = pd.read_csv('iris.csv', index_col=0)
```

```
In [4]: print(iris.shape)  
(150, 5)
```



# Line plot

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

```
Out[5]:
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

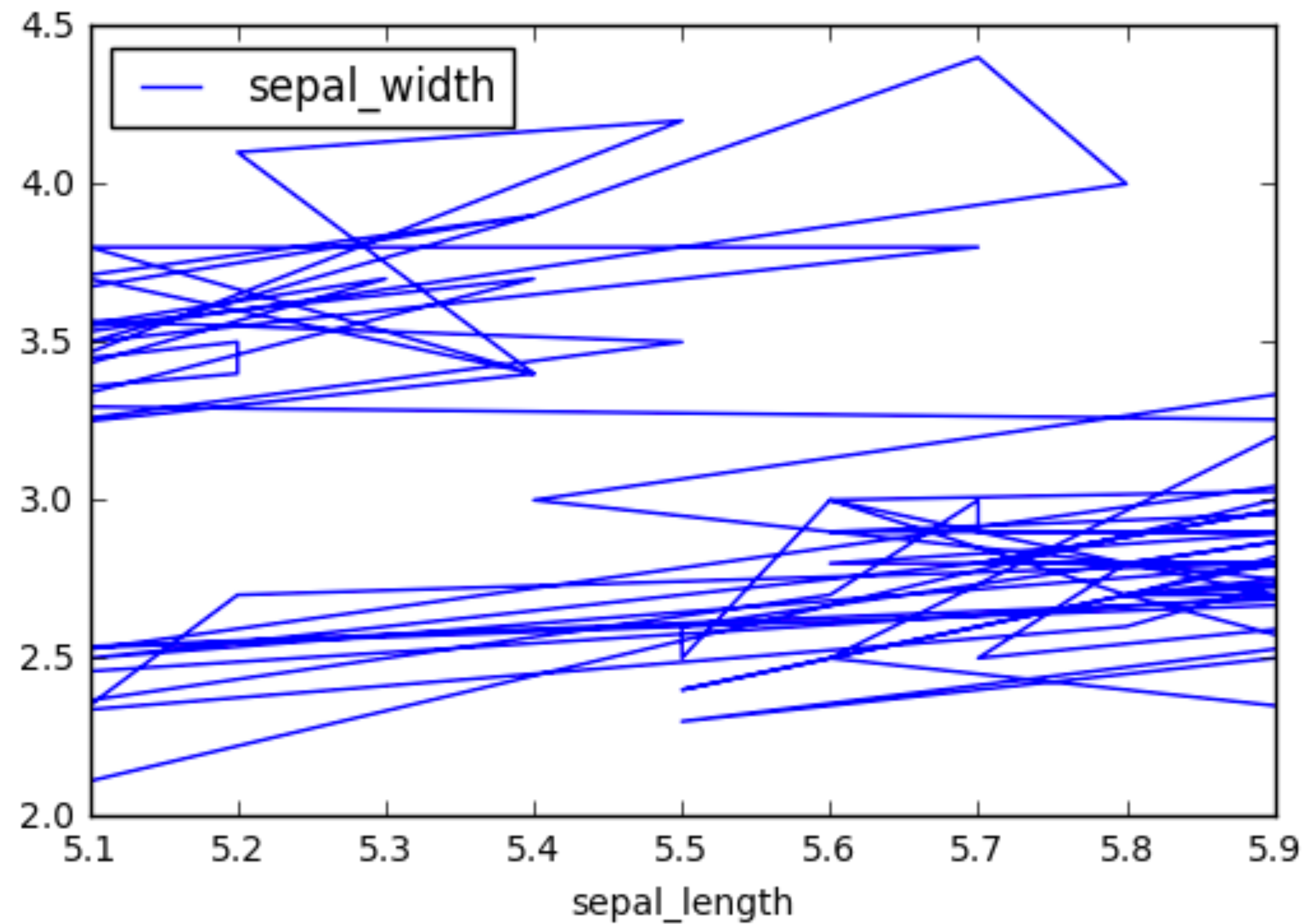
```
In [6]: iris.plot(x='sepal_length', y='sepal_width')
```

```
In [7]: plt.show()
```





# Line plot





# Scatter plot

```
In [8]: iris.plot(x='sepal_length', y='sepal_width',  
....:             kind='scatter')
```

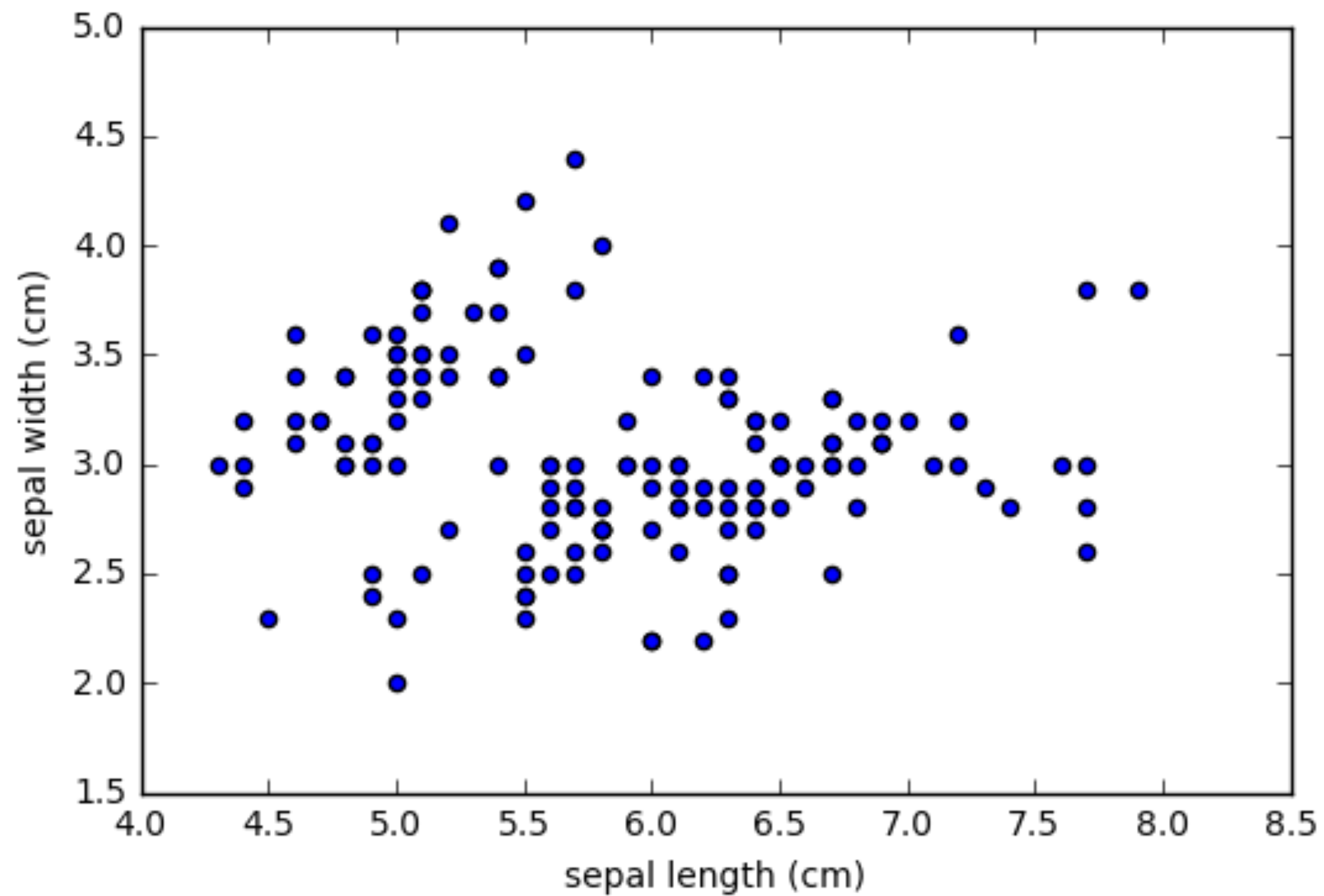
```
In [9]: plt.xlabel('sepal length (cm)')
```

```
In [10]: plt.ylabel('sepal width (cm)')
```

```
In [11]: plt.show()
```



# Scatter plot





# Box plot

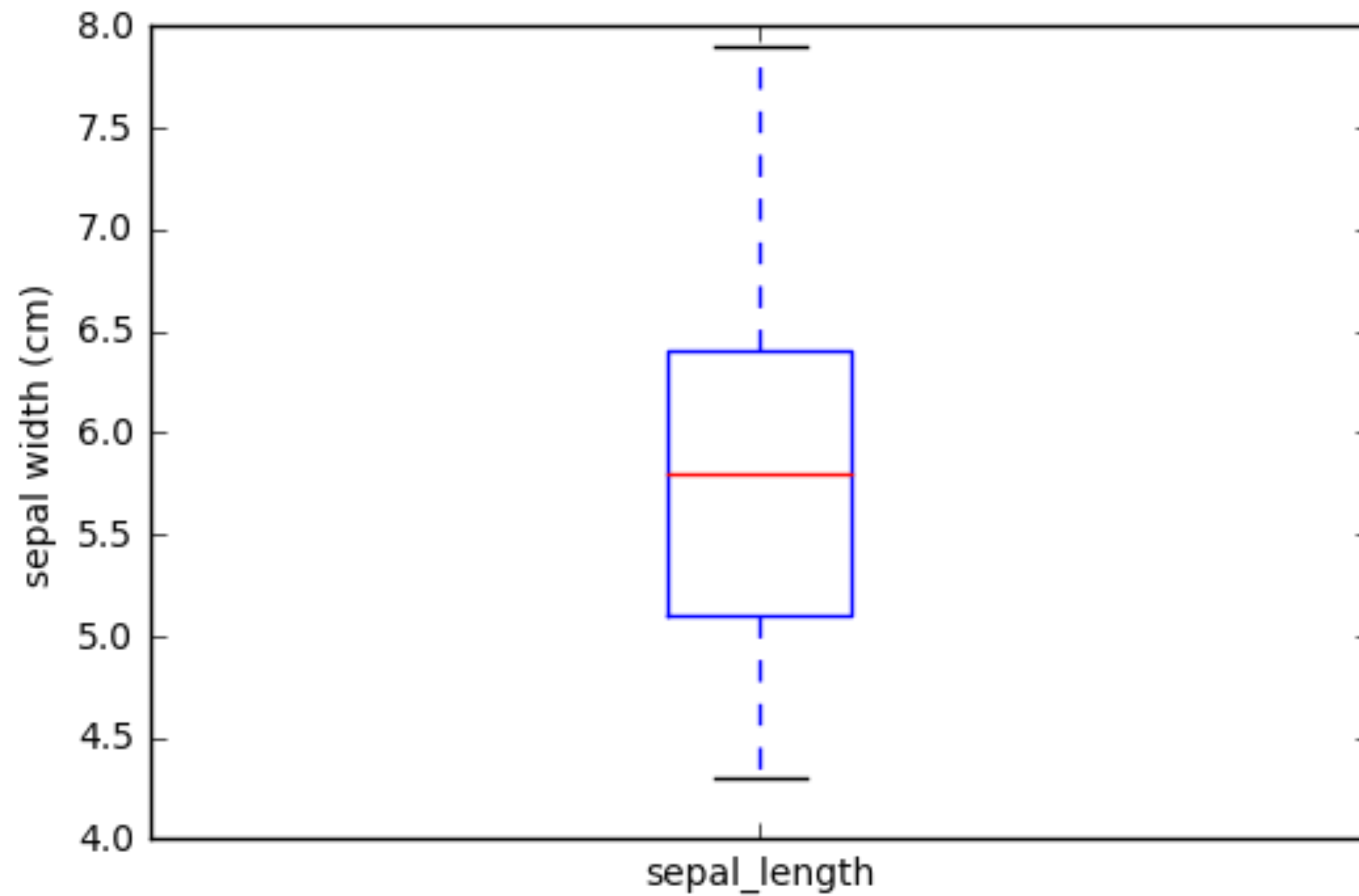
```
In [12]: iris.plot(y='sepal_length', kind='box')
```

```
In [13]: plt.ylabel('sepal width (cm)')
```

```
In [14]: plt.show()
```



# Box plot





# Histogram

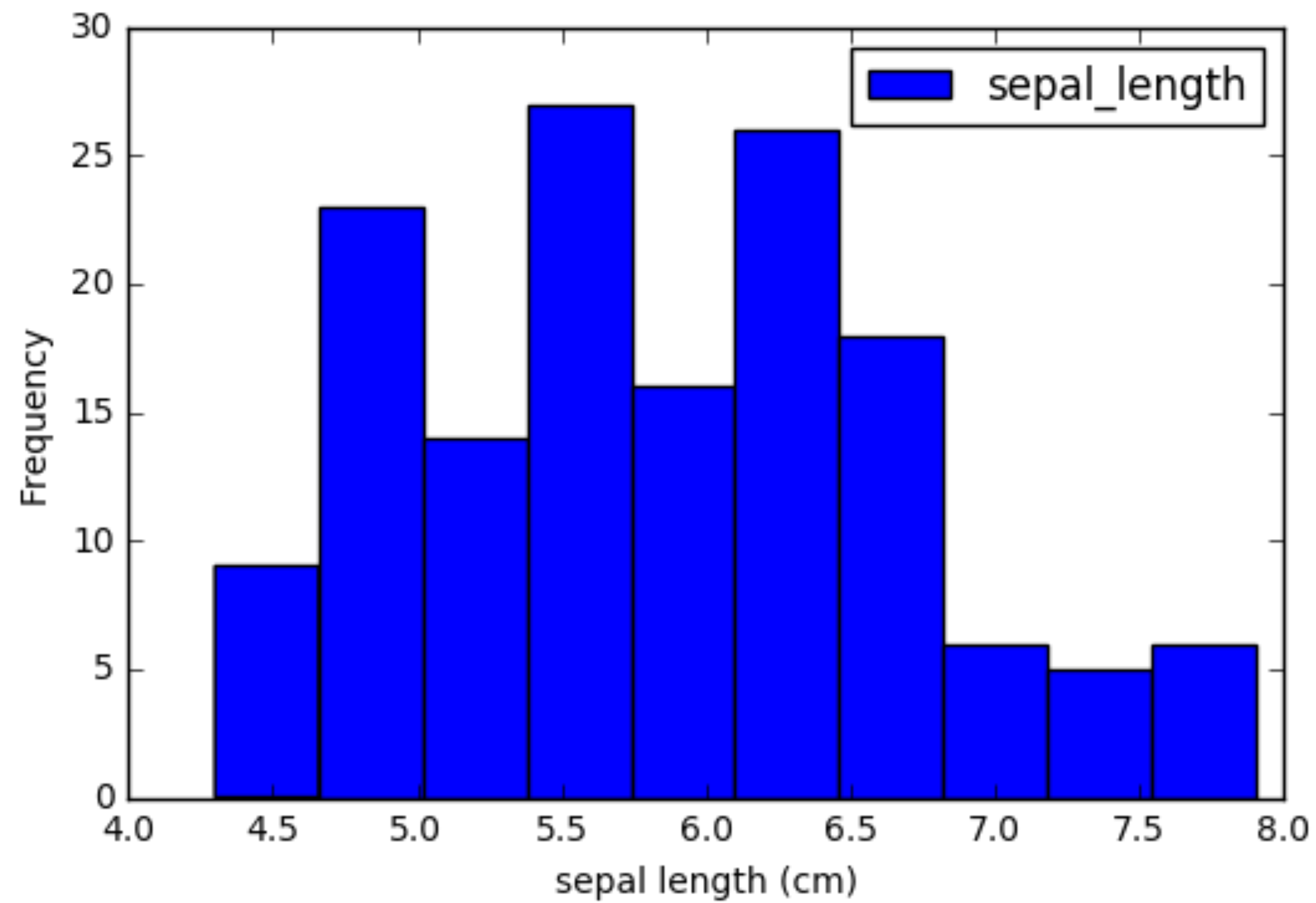
```
In [15]: iris.plot(y='sepal_length', kind='hist')
```

```
In [16]: plt.xlabel('sepal length (cm)')
```

```
In [17]: plt.show()
```



# Histogram





# Histogram options

- *bins* (integer): number of intervals or bins
- *range* (tuple): extrema of bins (minimum, maximum)
- *normed* (boolean): whether to normalize to one
- *cumulative* (boolean): compute Cumulative Distribution Function (CDF)
- ... more Matplotlib customizations





# Customizing histogram

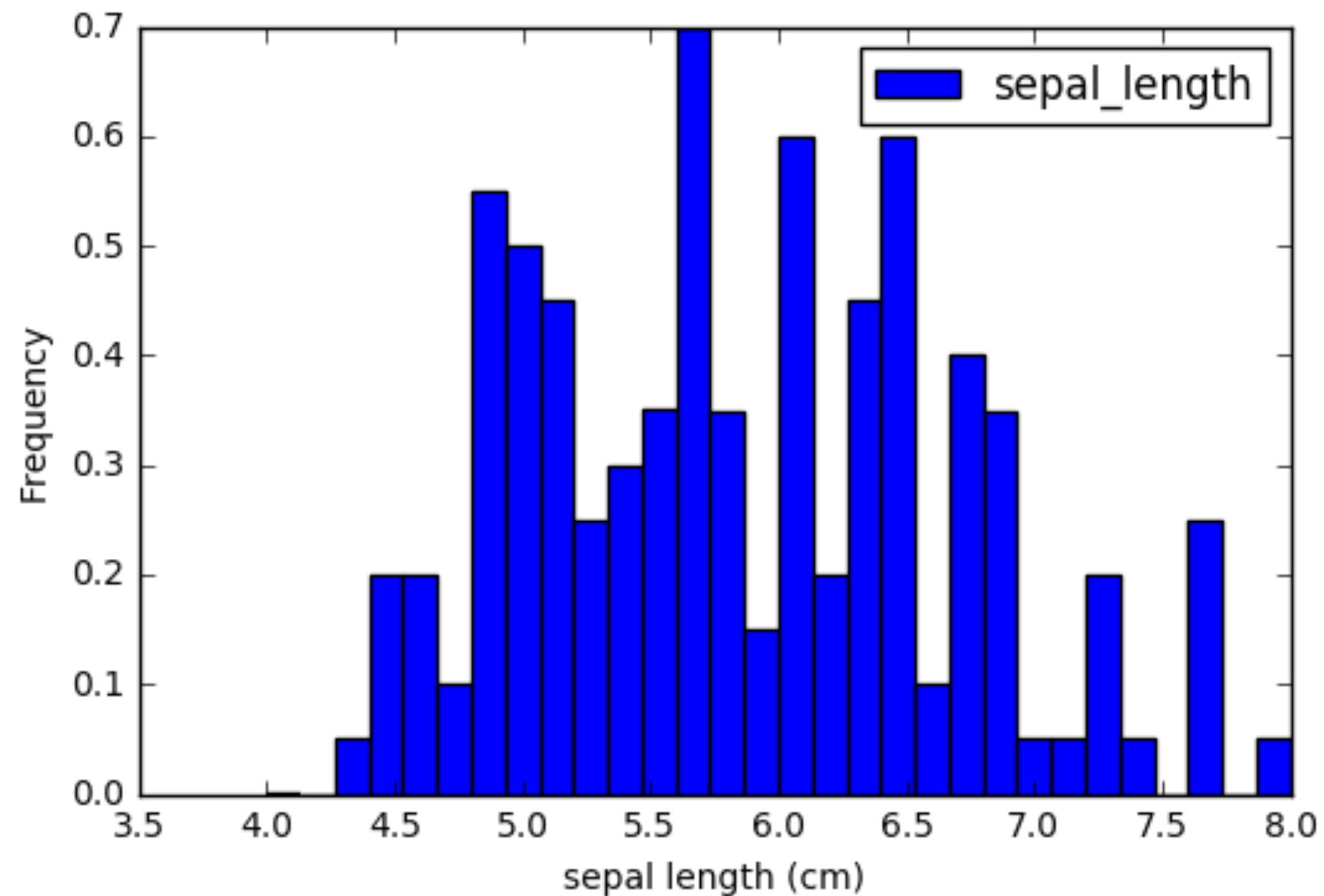
```
In [18]: iris.plot(y='sepal_length', kind='hist',  
....:             bins=30, range=(4,8), normed=True)
```

```
In [19]: plt.xlabel('sepal length (cm)')
```

```
In [20]: plt.show()
```



# Customizing histogram





# Cumulative distribution

```
In [21]: iris.plot(y='sepal_length', kind='hist', bins=30,  
....:               range=(4,8), cumulative=True, normed=True)
```

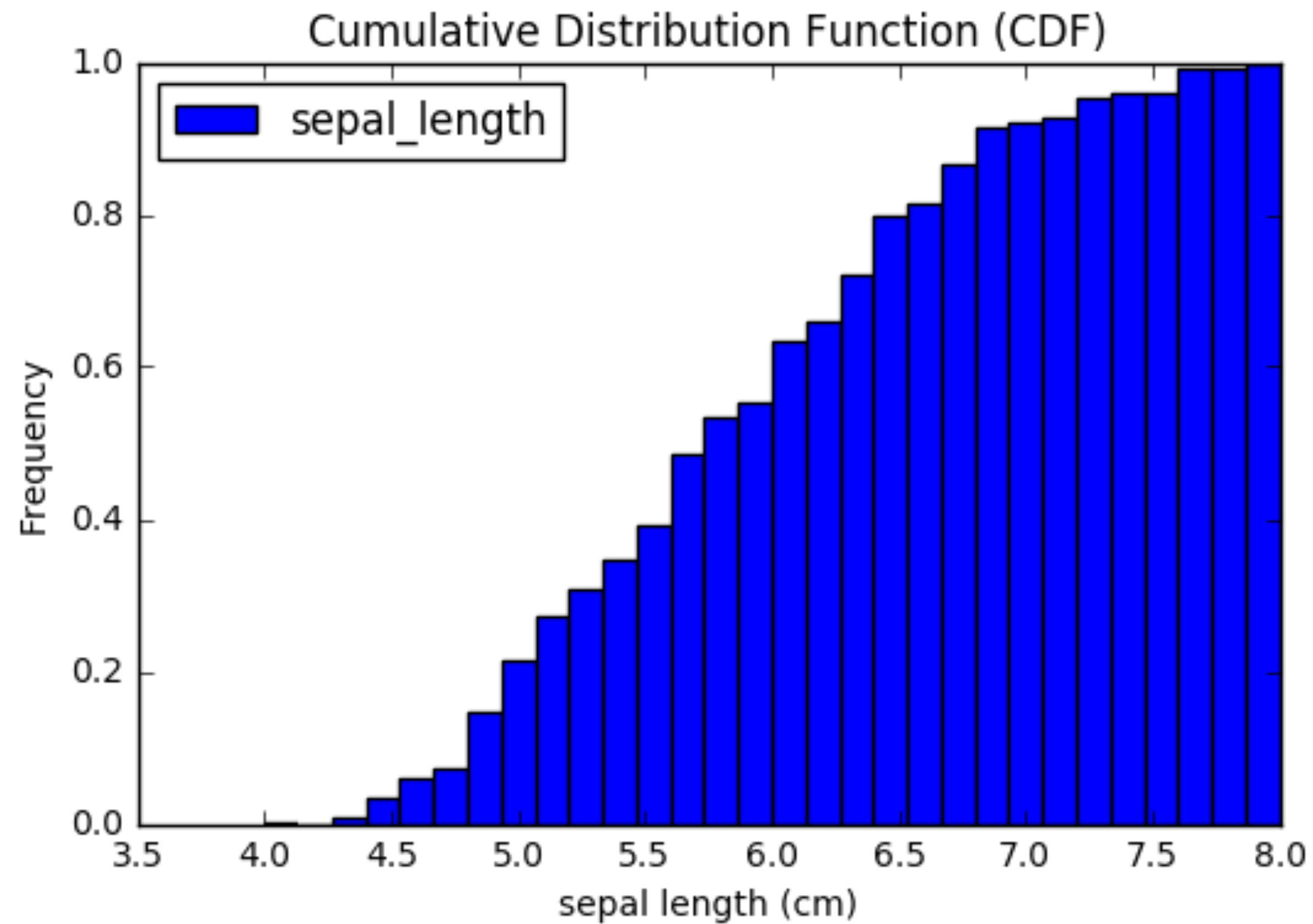
```
In [22]: plt.xlabel('sepal length (cm)')
```

```
In [23]: plt.title('Cumulative distribution function (CDF)')
```

```
In [24]: plt.show()
```



# Cumulative distribution



# Word of warning

- Three different DataFrame plot idioms
  - *iris.plot(kind='hist')*
  - *iris.plt.hist()*
  - *iris.hist()*
- Syntax/results differ!
- Pandas API still evolving: check documentation!



PANDAS FOUNDATIONS

**Let's practice!**



PANDAS FOUNDATIONS

# **Statistical exploratory data analysis**



# Summarizing with describe()

```
In [1]: iris.describe() # summary statistics
```

```
Out[1]:
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000





# Describe

- *count*: number of entries
- *mean*: average of entries
- *std*: standard deviation
- *min*: minimum entry
- *25%*: first quartile
- *50%*: median or second quartile
- *75%*: third quartile
- *max*: maximum entry



# Counts

```
In [2]: iris['sepal_length'].count() # Applied to Series
Out[2]: 150
```

```
In [3]: iris['sepal_width'].count() # Applied to Series
Out[3]: 150
```

```
In [4]: iris[['petal_length', 'petal_width']].count() # Applied
....: to DataFrame
Out[4]:
petal_length    150
petal_width     150
dtype: int64
```

```
In [5]: type(iris[['petal_length', 'petal_width']].count()) #
....: returns Series
Out[5]: pandas.core.series.Series
```



# Averages

```
In [6]: iris['sepal_length'].mean() # Applied to Series  
Out[6]: 5.8433333333333335
```

```
In [7]: iris.mean() # Applied to entire DataFrame  
Out[7]:  
sepal_length      5.843333  
sepal_width       3.057333  
petal_length      3.758000  
petal_width       1.199333  
dtype: float64
```

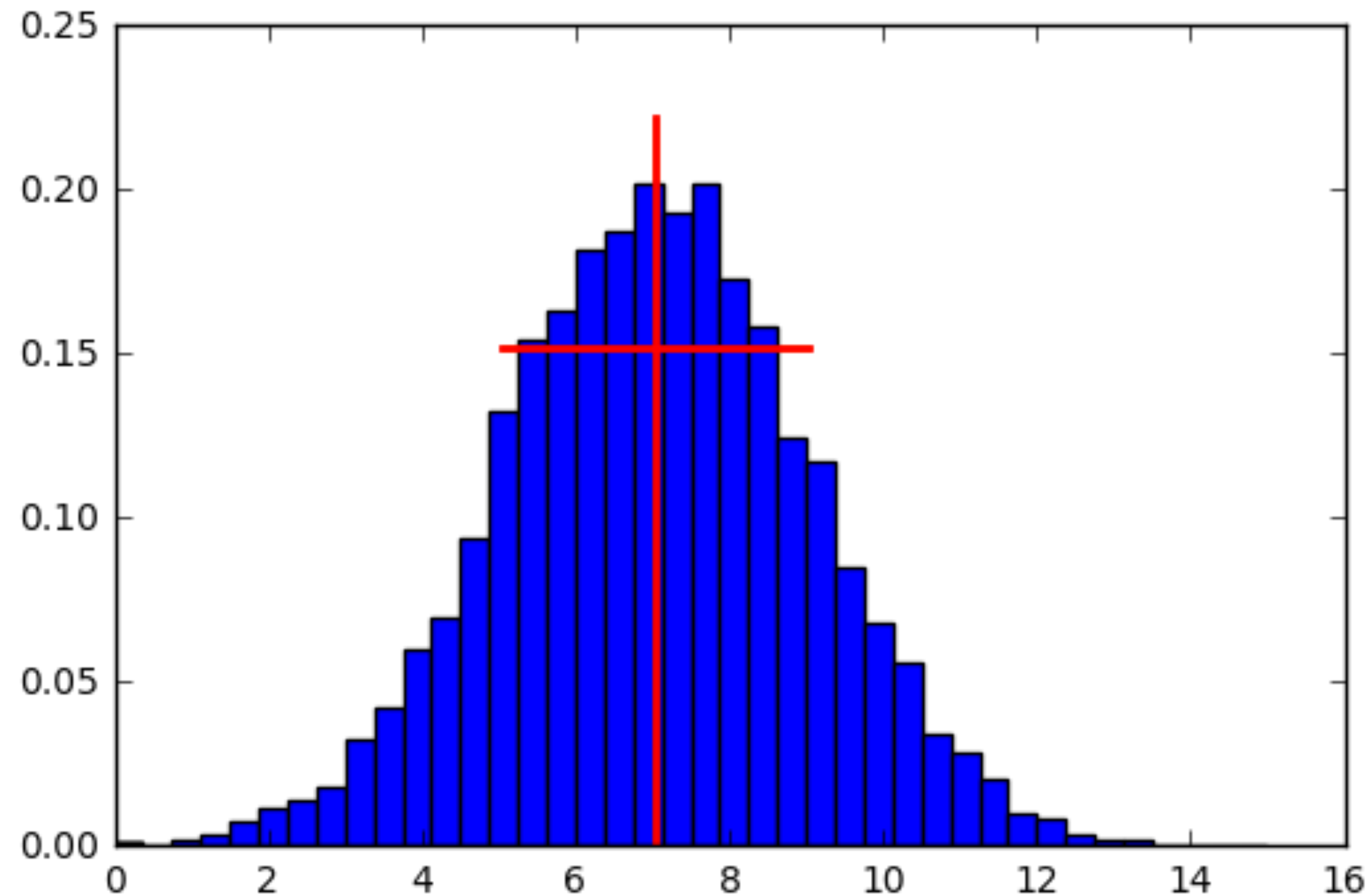


# Standard deviations

```
In [8]: iris.std()
Out[8]:
sepal_length    0.828066
sepal_width     0.435866
petal_length    1.765298
petal_width     0.762238
dtype: float64
```



# Mean and standard deviation on a bell curve



# Medians

```
In [9]: iris.median()  
Out[9]:  
sepal_length    5.80  
sepal_width     3.00  
petal_length    4.35  
petal_width     1.30  
dtype: float64
```



# Medians & 0.5 quantiles

```
In [10]: iris.median()
```

```
Out[10]:
```

```
sepal_length    5.80  
sepal_width     3.00  
petal_length    4.35  
petal_width     1.30  
dtype: float64
```

```
In [11]: q = 0.5
```

```
In [12]: iris.quantile(q)
```

```
Out[12]:
```

```
sepal_length    5.80  
sepal_width     3.00  
petal_length    4.35  
petal_width     1.30  
dtype: float64
```



# Inter-quartile range (IQR)

```
In [13]: q = [0.25, 0.75]
```

```
In [14]: iris.quantile(q)
```

```
Out[14]:
```

	sepal_length	sepal_width	petal_length	petal_width
0.25	5.1	2.8	1.6	0.3
0.75	6.4	3.3	5.1	1.8





# Ranges

```
In [15]: iris.min()
```

```
Out[15]:
```

```
sepal_length    4.3
sepal_width      2
petal_length     1
petal_width     0.1
species         setosa
dtype: object
```

```
In [16]: iris.max()
```

```
Out[16]:
```

```
sepal_length    7.9
sepal_width     4.4
petal_length     6.9
petal_width     2.5
species         virginica
dtype: object
```



# Box plots

```
In [17]: iris.plot(kind= 'box')
```

```
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x118a3d5f8>
```

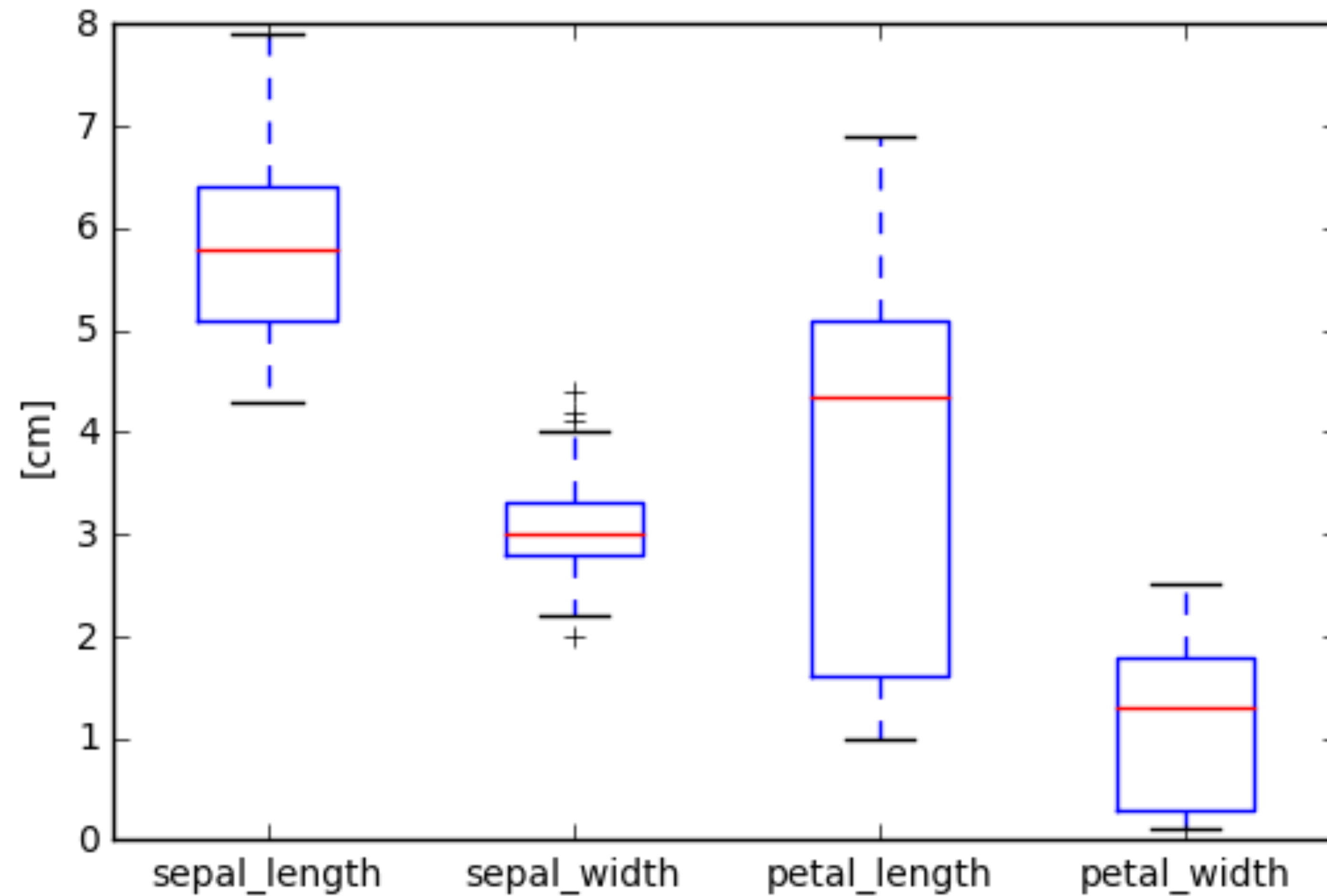
```
In [18]: plt.ylabel(' [cm] ')
```

```
Out[18]: <matplotlib.text.Text at 0x118a524e0>
```

```
In [19]: plt.show()
```



# Box plots





# Percentiles as quantiles

```
In [20]: iris.describe() # summary statistics
```

```
Out[20]:
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000



PANDAS FOUNDATIONS

**Let's practice!**



PANDAS FOUNDATIONS

# Separating populations



```
In [1]: iris.head()
```

```
Out[1]:
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa



# Describe species column

```
In [2]: iris['species'].describe()
```

```
Out[2]:
```

```
count          150  
unique           3  
top          setosa  
freq           50
```

```
Name: species, dtype: object
```

**count: # non-null entries**

**unique: # distinct values**

**top: most frequent category**

**freq: # occurrences of top**





# Unique & factors

```
In [3]: iris['species'].unique()  
Out[3]: array(['setosa', 'versicolor', 'virginica'], dtype=object)
```



# Filtering by species

```
In [4]: indices = iris['species'] == 'setosa'
```

```
In [5]: setosa = iris.loc[indices,:] # extract new DataFrame
```

```
In [6]: indices = iris['species'] == 'versicolor'
```

```
In [7]: versicolor = iris.loc[indices,:] # extract new DataFrame
```

```
In [8]: indices = iris['species'] == 'virginica'
```

```
In [9]: virginica = iris.loc[indices,:] # extract new DataFrame
```



# Checking species

```
In [10]: setosa['species'].unique()  
Out[10]: array(['setosa'], dtype=object)
```

```
In [11]: versicolor['species'].unique()  
Out[11]: array(['versicolor'], dtype=object)
```

```
In [12]: virginica['species'].unique()  
Out[12]: array(['virginica'], dtype=object)
```

```
In [13]: del setosa['species'], versicolor['species'],  
....:     virginica['species']
```



# Checking indexes

```
In [14]: setosa.head(2)
```

```
Out[14]:
```

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2

```
In [15]: versicolor.head(2)
```

```
Out[15]:
```

	sepal_length	sepal_width	petal_length	petal_width
50	7.0	3.2	4.7	1.4
51	6.4	3.2	4.5	1.5

```
In [16]: virginica.head(2)
```

```
Out[16]:
```

	sepal_length	sepal_width	petal_length	petal_width
100	6.3	3.3	6.0	2.5
101	5.8	2.7	5.1	1.9



# Visual EDA: all data

```
In [17]: iris.plot(kind= 'hist', bins=50, range=(0,8), alpha=0.3)
```

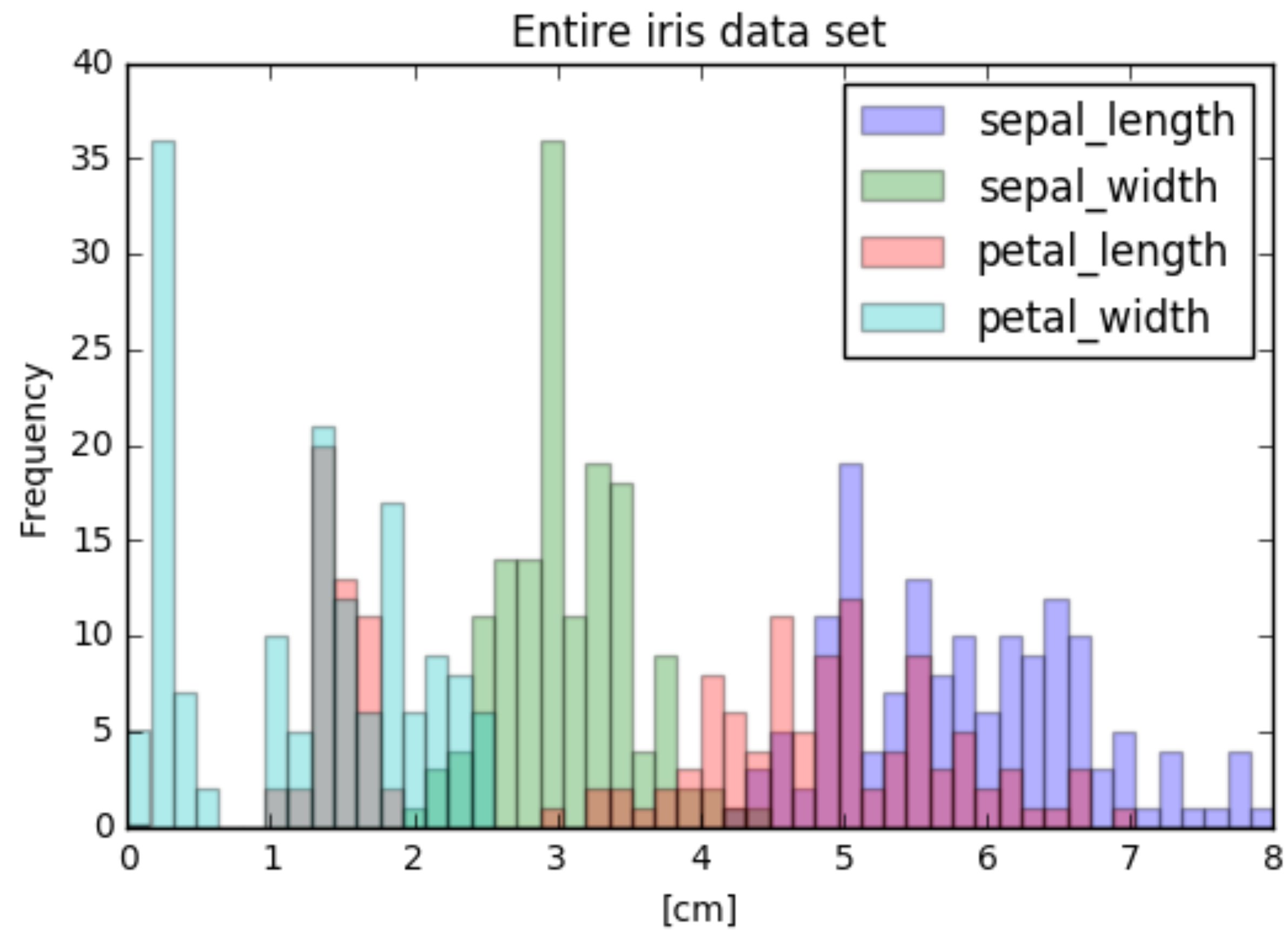
```
In [18]: plt.title('Entire iris data set')
```

```
In [19]: plt.xlabel(' [cm] ')
```

```
In [20]: plt.show()
```



# Visual EDA: all data





# Visual EDA: individual factors

```
In [21]: setosa.plot(kind='hist', bins=50, range=(0,8), alpha=0.3)
```

```
In [22]: plt.title('Setosa data set')
```

```
In [23]: plt.xlabel('[cm]')
```

```
In [24]: versicolor.plot(kind='hist', bins=50, range=(0,8), alpha=0.3)
```

```
In [25]: plt.title('Versicolor data set')
```

```
In [26]: plt.xlabel('[cm]')
```

```
In [27]: virginica.plot(kind='hist', bins=50, range=(0,8), alpha=0.3)
```

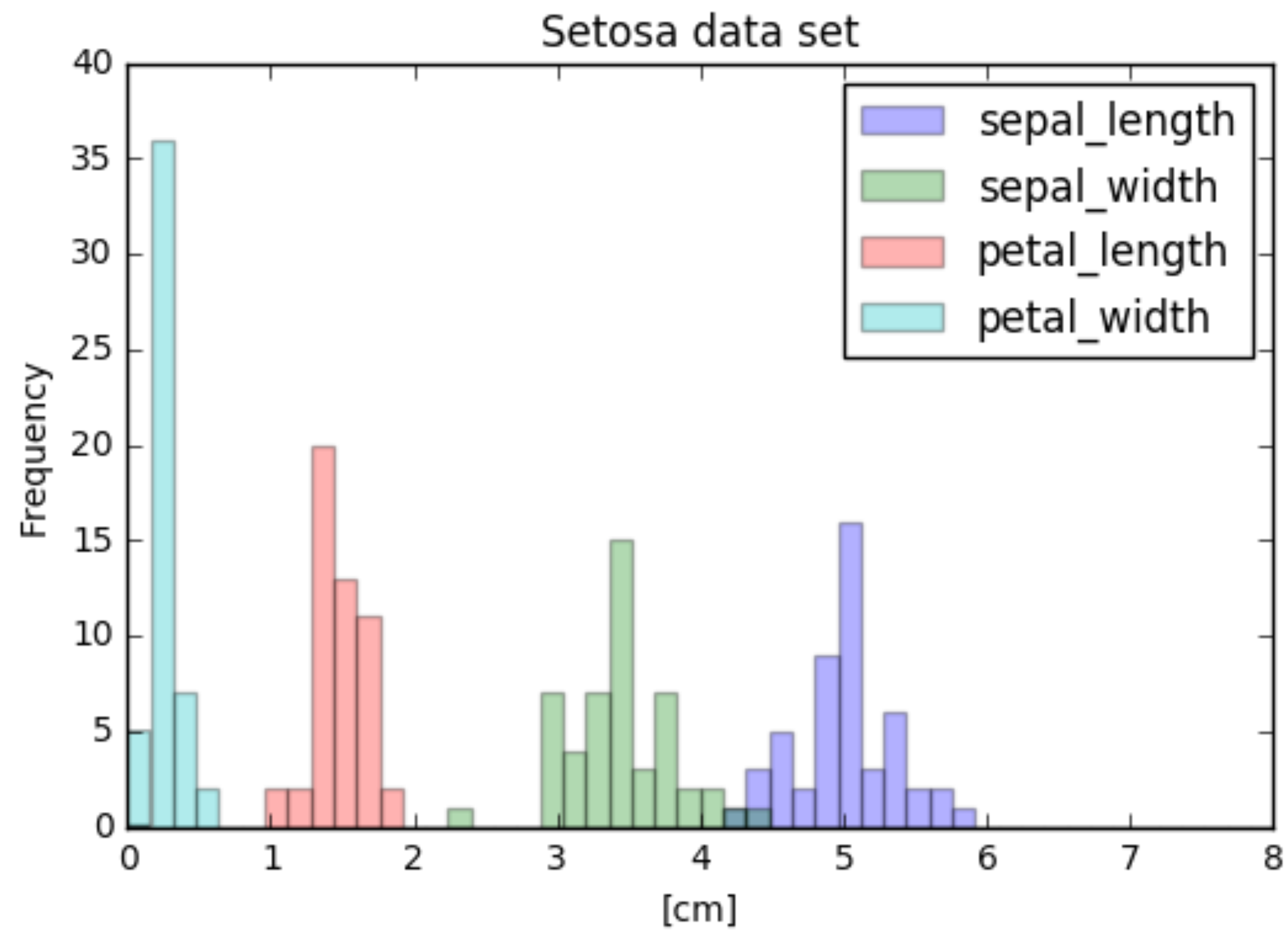
```
In [28]: plt.title('Virginica data set')
```

```
In [29]: plt.xlabel('[cm]')
```

```
In [30]: plt.show()
```



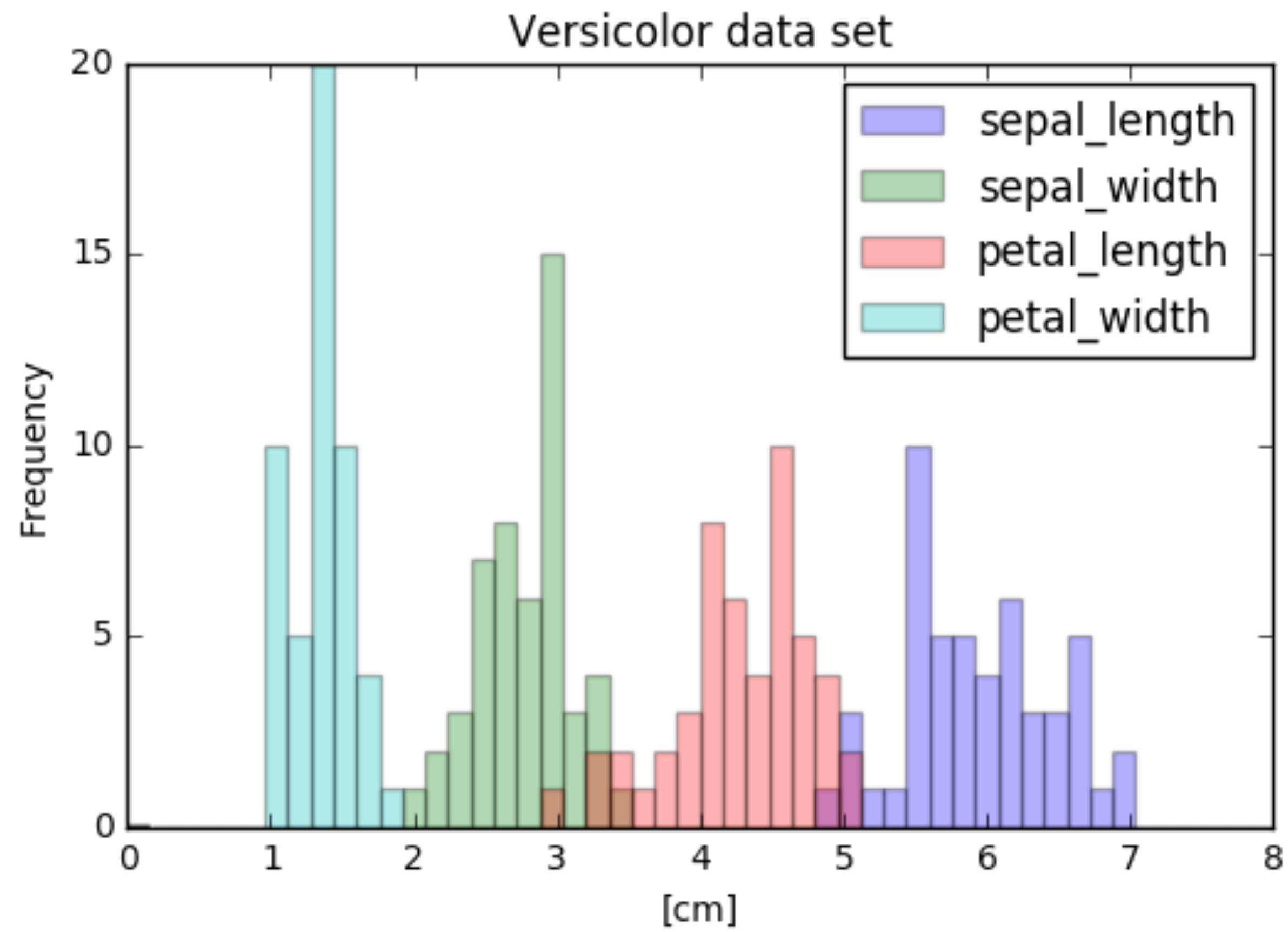
# Visual EDA: Setosa data





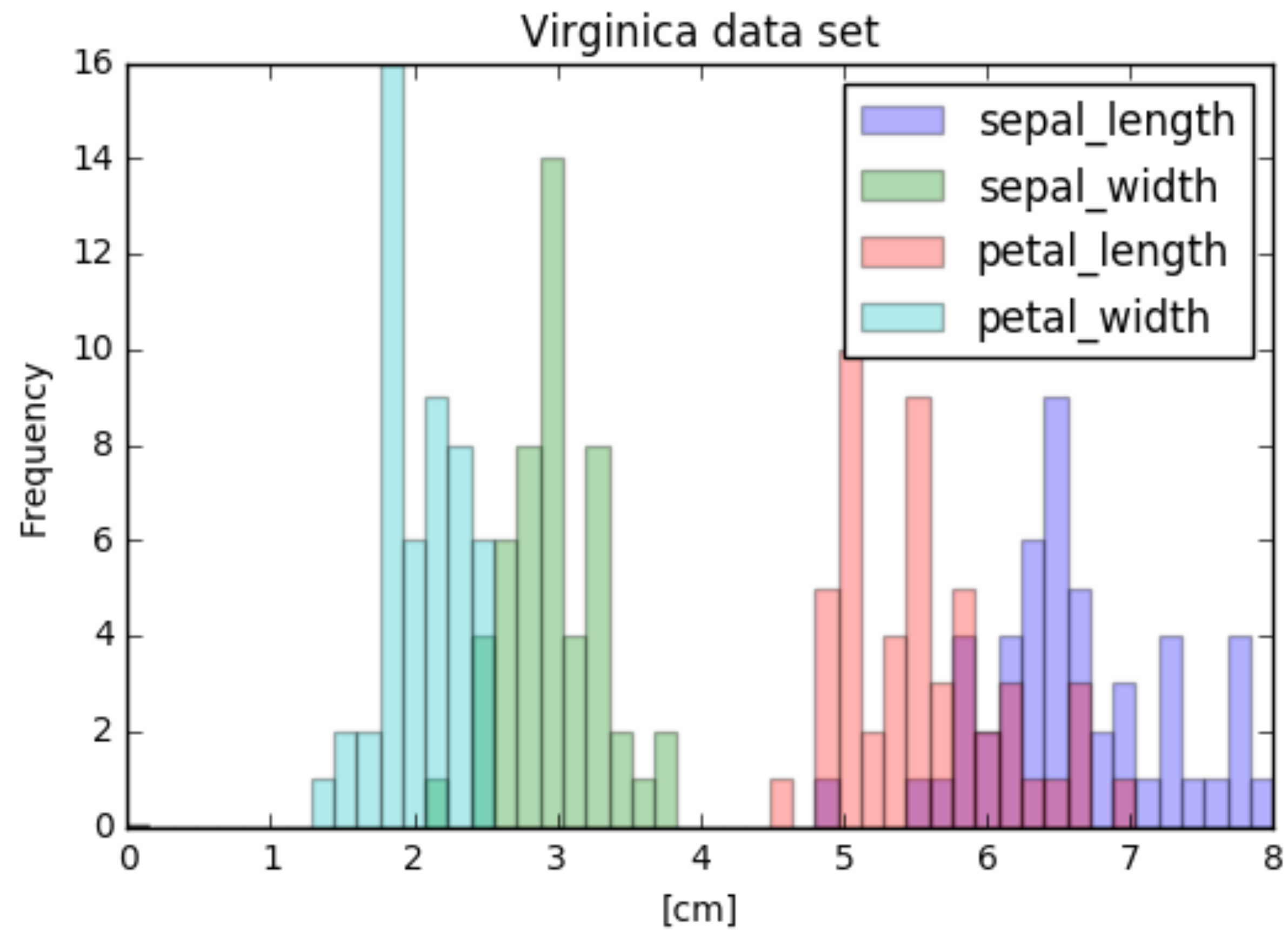


# Visual EDA: Versicolor data





# Visual EDA: Virginica data





# Statistical EDA: describe()

```
In [31]: describe_all = iris.describe()
```

```
In [32]: print(describe_all)
```

```
Out[32]:
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

```
In [33]: describe_setosa = setosa.describe()
```

```
In [34]: describe_versicolor = versicolor.describe()
```

```
In [35]: describe_virginica = virginica.describe()
```



# Computing errors

```
In [36]: error_setosa = 100 * np.abs(describe_setosa -  
    ...: describe_all)
```

```
In [37]: error_setosa = error_setosa/describe_setosa
```

```
In [38]: error_versicolor = 100 * np.abs(describe_versicolor -  
    ...: describe_all)
```

```
In [39]: error_versicolor = error_versicolor/describe_versicolor
```

```
In [40]: error_virginica = 100 * np.abs(describe_virginica -  
    ...: describe_all)
```

```
In [41]: error_virginica = error_virginica/describe_virginica
```



# Viewing errors

```
In [42]: print(error_setosa)
```

	sepal_length	sepal_width
count	200.000000	200.000000
mean	16.726595	10.812913
std	134.919250	14.984768
min	0.000000	13.043478
25%	6.250000	12.500000
50%	16.000000	11.764706
75%	23.076923	10.204082
max	36.206897	0.000000

petal_length
200.000000
157.045144
916.502136
0.000000
14.285714
190.000000
223.809524
263.157895

petal_width
200.000000
387.533875
623.284534
0.000000
50.000000
550.000000
500.000000
316.666667



PANDAS FOUNDATIONS

**Let's practice!**



PANDAS FOUNDATIONS

# Indexing time series



# Using pandas to read datetime objects

- `read_csv()` function
  - Can read strings into datetime objects
  - Need to specify `'parse_dates=True'`
- ISO 8601 format
  - `yyyy-mm-dd hh:mm:ss`





# Product sales CSV

	Date	Company	Product	Units
0	2015-02-02 08:30:00	Hooli	Software	3
1	2015-02-02 21:00:00	Mediacore	Hardware	9
2	2015-02-03 14:00:00	Initech	Software	13
3	2015-02-04 15:30:00	Streeplex	Software	13
4	2015-02-04 22:00:00	Acme Coporation	Hardware	14





# Parse dates

```
In [3]: sales.head()
```

```
Out[3]:
```

Date	Company	Product	Units
2015-02-02 08:30:00	Hooli	Software	3
2015-02-02 21:00:00	Mediacore	Hardware	9
2015-02-03 14:00:00	Initech	Software	13
2015-02-04 15:30:00	Streeplex	Software	13
2015-02-04 22:00:00	Acme Coporation	Hardware	14



# Parse dates

```
In [4]: sales.info()
DatetimeIndex: 19 entries, 2015-02-02 08:30:00 to 2015-02-26
09:00:00
Data columns (total 3 columns):
Company      19 non-null object
Product      19 non-null object
Units        19 non-null int64
dtypes: int64(1), object(2)
memory usage: 608.0+ bytes
```



# Selecting single datetime

```
In [5]: sales.loc['2015-02-19 11:00:00', 'Company']  
Out[5]: 'Mediacore'
```



# Selecting whole day

```
In [6]: sales.loc['2015-2-5']
```

```
Out[6]:
```

Date		Company	Product	Units
2015-02-05 02:00:00	Acme	Coporation	Software	19
2015-02-05 22:00:00		Hooli	Service	10

# Partial datetime string selection

- Alternative formats:
  - `sales.loc['February 5, 2015']`
  - `sales.loc['2015-Feb-5']`
- Whole month: `sales.loc['2015-2']`
- Whole year: `sales.loc['2015']`



# Selecting whole month

```
In [7]: sales.loc['2015-2']  
Out[7]:
```

Date	Company	Product	Units
2015-02-02 08:30:00	Hooli	Software	3
2015-02-02 21:00:00	Mediacore	Hardware	9
2015-02-03 14:00:00	Initech	Software	13
2015-02-04 15:30:00	Streeplex	Software	13
2015-02-04 22:00:00	Acme Coporation	Hardware	14
2015-02-05 02:00:00	Acme Coporation	Software	19
2015-02-05 22:00:00	Hooli	Service	10
2015-02-07 23:00:00	Acme Coporation	Hardware	1
2015-02-09 09:00:00	Streeplex	Service	19
2015-02-09 13:00:00	Mediacore	Software	7
2015-02-11 20:00:00	Initech	Software	7
2015-02-11 23:00:00	Hooli	Software	4
2015-02-16 12:00:00	Hooli	Software	10
2015-02-19 11:00:00	Mediacore	Hardware	16
...			





# Slicing using dates/times

```
In [8]: sales.loc['2015-2-16':'2015-2-20']
```

```
Out[8]:
```

Date	Company	Product	Units
2015-02-16 12:00:00	Hooli	Software	10
2015-02-19 11:00:00	Mediacore	Hardware	16
2015-02-19 16:00:00	Mediacore	Service	10



# Convert strings to datetime

```
In [9]: evening_2_11 = pd.to_datetime(['2015-2-11 20:00',  
...: '2015-2-11 21:00', '2015-2-11 22:00', '2015-2-11 23:00'])
```

```
In [10]: evening_2_11
```

```
Out[10]:
```

```
DatetimeIndex(['2015-02-11 20:00:00', '2015-02-11 21:00:00',  
               '2015-02-11 22:00:00', '2015-02-11 23:00:00'],  
              dtype='datetime64[ns]', freq=None)
```



# Reindexing DataFrame

```
In [11]: sales.reindex(evening_2_11)
```

```
Out[11]:
```

		Company	Product	Units
2015-02-11	20:00:00	Initech	Software	7.0
2015-02-11	21:00:00	NaN	NaN	NaN
2015-02-11	22:00:00	NaN	NaN	NaN
2015-02-11	23:00:00	Hooli	Software	4.0



# Filling missing values

```
In [12]: sales.reindex(evening_2_11, method='ffill')  
Out[12]:
```

		Company	Product	Units
2015-02-11	20:00:00	Initech	Software	7
2015-02-11	21:00:00	Initech	Software	7
2015-02-11	22:00:00	Initech	Software	7
2015-02-11	23:00:00	Hooli	Software	4

```
In [13]: sales.reindex(evening_2_11, method='bfill')  
Out[13]:
```

		Company	Product	Units
2015-02-11	20:00:00	Initech	Software	7
2015-02-11	21:00:00	Hooli	Software	4
2015-02-11	22:00:00	Hooli	Software	4
2015-02-11	23:00:00	Hooli	Software	4



PANDAS FOUNDATIONS

**Let's practice!**



PANDAS FOUNDATIONS

# **Resampling time series data**



# Sales data

```
In [1]: import pandas as pd
```

```
In [2]: sales = pd.read_csv('sales-feb-2015.csv',  
    ....:                   parse_dates=True, index_col= 'Date')
```

```
In [3]: sales.head()
```

```
Out[3]:
```

		Company	Product	Units
Date				
2015-02-02 08:30:00		Hooli	Software	3
2015-02-02 21:00:00		Mediacore	Hardware	9
2015-02-03 14:00:00		Initech	Software	13
2015-02-04 15:30:00		Streeplex	Software	13
2015-02-04 22:00:00	Acme	Coporation	Hardware	14



# Resampling

- Statistical methods over different time intervals
  - `mean()`, `sum()`, `count()`, etc.
- Down-sampling
  - reduce datetime rows to slower frequency
- Up-sampling
  - increase datetime rows to faster frequency





# Aggregating means

```
In [4]: daily_mean = sales.resample('D').mean()
```

```
In [5]: daily_mean
```

```
Out[5]:
```

	Units
Date	
2015-02-02	6.0
2015-02-03	13.0
2015-02-04	13.5
2015-02-05	14.5
2015-02-06	NaN
2015-02-07	1.0
2015-02-08	NaN
2015-02-09	13.0
2015-02-10	NaN
2015-02-11	5.5
2015-02-12	NaN
2015-02-13	NaN
2015-02-14	NaN



# Verifying

```
In [6]: print(daily_mean.loc['2015-2-2'])  
Units      6.0  
Name: 2015-02-02 00:00:00, dtype: float64
```

```
In [7]: print(sales.loc['2015-2-2', 'Units'])  
Date  
2015-02-02 08:30:00      3  
2015-02-02 21:00:00      9  
Name: Units, dtype: int64
```

```
In [8]: sales.loc['2015-2-2', 'Units'].mean()  
Out[8]: 6.0
```



# Method chaining

```
In [9]: sales.resample('D').sum()
```

```
Out[9]:
```

	Units
Date	
2015-02-02	6.0
2015-02-03	13.0
2015-02-04	13.5
2015-02-05	14.5
2015-02-06	NaN
2015-02-07	1.0
2015-02-08	NaN
2015-02-09	13.0
2015-02-10	NaN
2015-02-11	5.5
2015-02-12	NaN
2015-02-13	NaN

# Method chaining

```
In [10]: sales.resample('D').sum().max()  
Out[10]:  
Units      29.0  
dtype: float64
```



# Resampling strings

```
In [11]: sales.resample('W').count()
```

```
Out[11]:
```

	Company	Product	Units
Date			
2015-02-08	8	8	8
2015-02-15	4	4	4
2015-02-22	5	5	5
2015-03-01	2	2	2



# Resampling frequencies

Input	Description
'min', 'T'	minute
'H'	hour
'D'	day
'B'	business day
'W'	week
'M'	month
'Q'	quarter
'A'	year



# Multiplying frequencies

```
In [12]: sales.loc[:, 'Units'].resample('2W').sum()  
Out[12]:  
Date  
2015-02-08      82  
2015-02-22      79  
2015-03-08      14  
Freq: 2W-SUN, Name: Units, dtype: int64
```



# Upsampling

```
In [13]: two_days = sales.loc['2015-2-4': '2015-2-5', 'Units']
```

```
In [13]: two_days
```

```
Out[13]:
```

```
Date
```

```
2015-02-04 15:30:00    13
```

```
2015-02-04 22:00:00    14
```

```
2015-02-05 02:00:00    19
```

```
2015-02-05 22:00:00    10
```

```
Name: Units, dtype: int64
```





# Upsampling and filling

```
In [14]: two_days.resample('4H').ffill()
```

```
Out[14]:
```

```
Date
```

```
Date
```

```
2015-02-04 12:00:00      NaN
```

```
2015-02-04 16:00:00    13.0
```

```
2015-02-04 20:00:00    13.0
```

```
2015-02-05 00:00:00    14.0
```

```
2015-02-05 04:00:00    19.0
```

```
2015-02-05 08:00:00    19.0
```

```
2015-02-05 12:00:00    19.0
```

```
2015-02-05 16:00:00    19.0
```

```
2015-02-05 20:00:00    19.0
```

```
Freq: 4H, Name: Units, dtype: float64
```



PANDAS FOUNDATIONS

**Let's practice!**



PANDAS FOUNDATIONS

# **Manipulating time series data**



# Sales data

```
In [1]: import pandas as pd
```

```
In [2]: sales = pd.read_csv('sales-feb-2015.csv',  
    ....:                  parse_dates=['Date'])
```

```
In [3]: sales.head()
```

```
Out[3]:
```

	Date	Company	Product	Units
0	2015-02-02 08:30:00	Hooli	Software	3
1	2015-02-02 21:00:00	Mediacore	Hardware	9
2	2015-02-03 14:00:00	Initech	Software	13
3	2015-02-04 15:30:00	Streeplex	Software	13
4	2015-02-04 22:00:00	Acme Coporation	Hardware	14



# String methods

```
In [4]: sales['Company'].str.upper()
```

```
Out[4]:
```

```
0          HOOLI
1      MEDIACORE
2      INITECH
3      STREEPLEX
4  ACME COPORATION
5  ACME COPORATION
6          HOOLI
7  ACME COPORATION
8      STREEPLEX
9      MEDIACORE
10         INITECH
11         HOOLI
12         HOOLI
13      MEDIACORE
14      MEDIACORE
15      MEDIACORE
```

```
...
```



# Substring matching

```
In [5]: sales['Product'].str.contains('ware')
```

```
Out[5]:
```

```
0      True
1      True
2      True
3      True
4      True
5      True
6     False
7      True
8     False
9      True
10     True
11     True
12     True
13     True
14     False
...
```



# Boolean arithmetic

```
In [6]: True + False
```

```
Out[6]: 1
```

```
In [7]: True + True
```

```
Out[7]: 2
```

```
In [8]: False + False
```

```
Out[8]: 0
```



# Boolean reduction

```
In [9]: sales['Product'].str.contains('ware').sum()  
Out[9]: 14
```





# Datetime methods

```
In [9]: sales['Date'].dt.hour
```

```
Out[9]:
```

```
0      8
1     21
2     14
3     15
4     22
5      2
6     22
7     23
8      9
9     13
10    20
11    23
12    12
13    11
14    16
...
```



# Set timezone

```
In [10]: central = sales['Date'].dt.tz_localize('US/Central')
```

```
In [11]: central
```

```
Out[11]:
```

```
0      2015-02-02 08:30:00-06:00
1      2015-02-02 21:00:00-06:00
2      2015-02-03 14:00:00-06:00
3      2015-02-04 15:30:00-06:00
4      2015-02-04 22:00:00-06:00
5      2015-02-05 02:00:00-06:00
6      2015-02-05 22:00:00-06:00
7      2015-02-07 23:00:00-06:00
8      2015-02-09 09:00:00-06:00
9      2015-02-09 13:00:00-06:00
10     2015-02-11 20:00:00-06:00
11     2015-02-11 23:00:00-06:00
12     2015-02-16 12:00:00-06:00
```

```
...
```

```
Name: Date, dtype: datetime64[ns, US/Central]
```



# Convert timezone

```
In [12]: central.dt.tz_convert('US/Eastern')
```

```
Out[12]:
```

```
0    2015-02-02 09:30:00-05:00
1    2015-02-02 22:00:00-05:00
2    2015-02-03 15:00:00-05:00
3    2015-02-04 16:30:00-05:00
4    2015-02-04 23:00:00-05:00
5    2015-02-05 03:00:00-05:00
6    2015-02-05 23:00:00-05:00
7    2015-02-08 00:00:00-05:00
8    2015-02-09 10:00:00-05:00
9    2015-02-09 14:00:00-05:00
10   2015-02-11 21:00:00-05:00
11   2015-02-12 00:00:00-05:00
12   2015-02-16 13:00:00-05:00
13   2015-02-19 12:00:00-05:00
14   2015-02-19 17:00:00-05:00
```

```
...
```

```
Name: Date, dtype: datetime64[ns, US/Eastern]
```



# Method chaining

```
In [13]: sales['Date'].dt.tz_localize('US/Central').  
        ...: dt.tz_convert('US/Eastern')
```

```
Out[13]:
```

```
0      2015-02-02 09:30:00-05:00  
1      2015-02-02 22:00:00-05:00  
2      2015-02-03 15:00:00-05:00  
3      2015-02-04 16:30:00-05:00  
4      2015-02-04 23:00:00-05:00  
5      2015-02-05 03:00:00-05:00  
6      2015-02-05 23:00:00-05:00  
7      2015-02-08 00:00:00-05:00  
8      2015-02-09 10:00:00-05:00  
9      2015-02-09 14:00:00-05:00  
10     2015-02-11 21:00:00-05:00  
11     2015-02-12 00:00:00-05:00  
12     2015-02-16 13:00:00-05:00  
13     2015-02-19 12:00:00-05:00  
14     2015-02-19 17:00:00-05:00
```

```
...
```

```
Name: Date, dtype: datetime64[ns, US/Eastern]
```



# World Population

```
In [14]: population = pd.read_csv('world_population.csv',  
    ...: parse_dates=True, index_col= 'Date')
```

```
In [15]: population
```

```
Out[15]:
```

Date	Population
1960-12-31	2.087485e+10
1970-12-31	2.536513e+10
1980-12-31	3.057186e+10
1990-12-31	3.644928e+10
2000-12-31	4.228550e+10
2010-12-31	4.802217e+10



# Upsample population

```
In [16]: population.resample('A').first()  
Out[16]:
```

	Population
Date	
1960-12-31	2.087485e+10
1961-12-31	NaN
1962-12-31	NaN
1963-12-31	NaN
1964-12-31	NaN
1965-12-31	NaN
1966-12-31	NaN
1967-12-31	NaN
1968-12-31	NaN
1969-12-31	NaN
1970-12-31	2.536513e+10
1971-12-31	NaN
1972-12-31	NaN



# Interpolate missing data

```
In [17]: population.resample('A').first().interpolate('linear')  
Out[17]:
```

Date	Population
1960-12-31	2.087485e+10
1961-12-31	2.132388e+10
1962-12-31	2.177290e+10
1963-12-31	2.222193e+10
1964-12-31	2.267096e+10
1965-12-31	2.311999e+10
1966-12-31	2.356902e+10
1967-12-31	2.401805e+10
1968-12-31	2.446707e+10
1969-12-31	2.491610e+10
1970-12-31	2.536513e+10
1971-12-31	2.588580e+10
1972-12-31	2.640648e+10



PANDAS FOUNDATIONS

**Let's practice!**





PANDAS FOUNDATIONS

# **Time series visualization**



# Topics

- Line types
- Plot types
- Subplots



# S&P 500 Data

```
In [1]: import pandas as pd
```

```
In [2]: import matplotlib.pyplot as plt
```

```
In [3]: sp500 = pd.read_csv('sp500.csv', parse_dates=True,  
....:                       index_col= 'Date')
```

```
In [4]: sp500.head()
```

```
Out[4]:
```

	Open	High	Low	Close	Volume	Adj Close
Date						
2010-01-04	1116.560059	1133.869995	1116.560059	1132.989990	3991400000	1132.989990
2010-01-05	1132.660034	1136.630005	1129.660034	1136.520020	2491020000	1136.520020
2010-01-06	1135.709961	1139.189941	1133.949951	1137.140015	4972660000	1137.140015
2010-01-07	1136.270020	1142.459961	1131.319946	1141.689941	5270680000	1141.689941
2010-01-08	1140.520020	1145.390015	1136.219971	1144.979980	4389590000	1144.979980



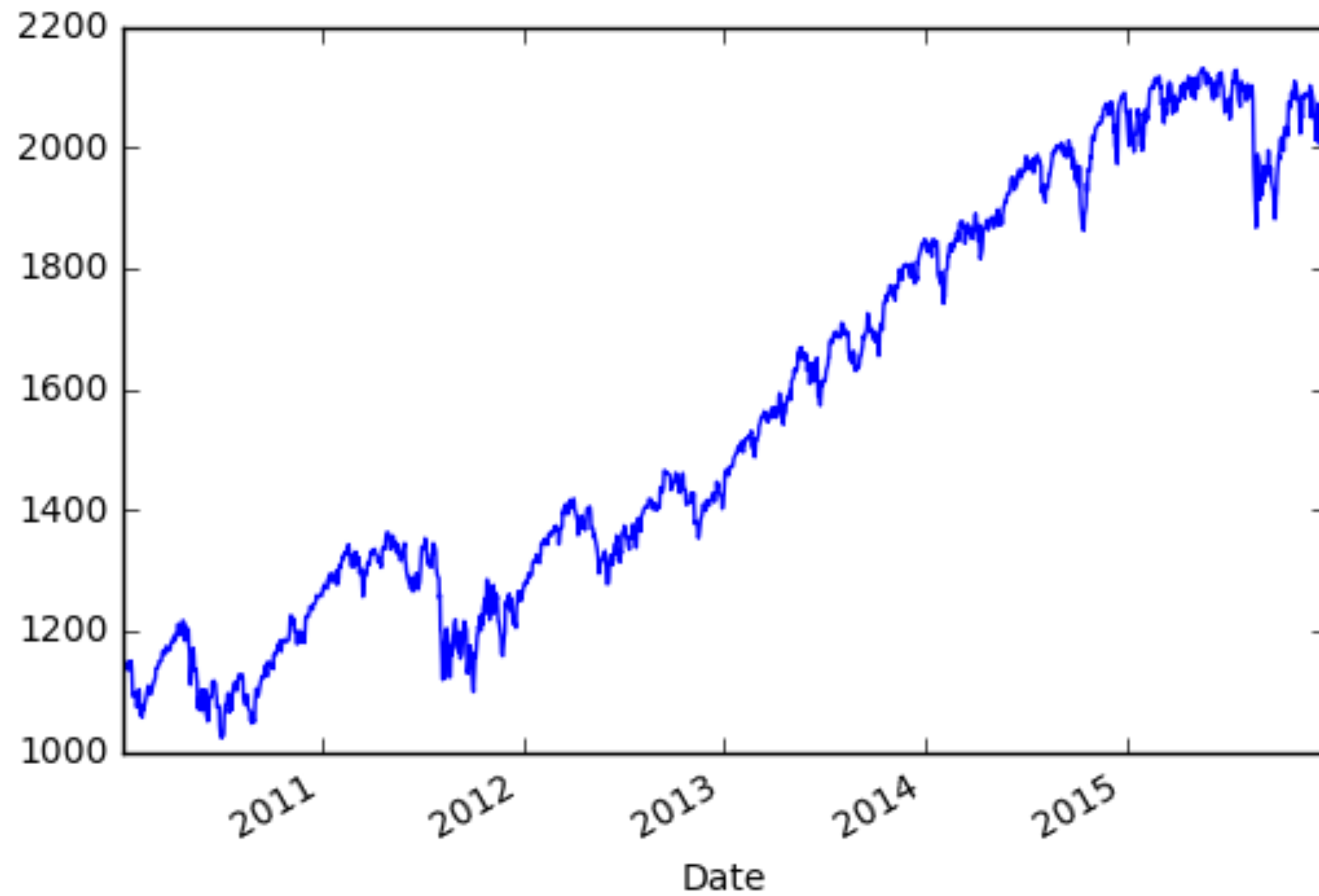
# Pandas plot

```
In [5]: sp500['Close'].plot()
```

```
In [6]: plt.show()
```



# Default plot





# Labels and title

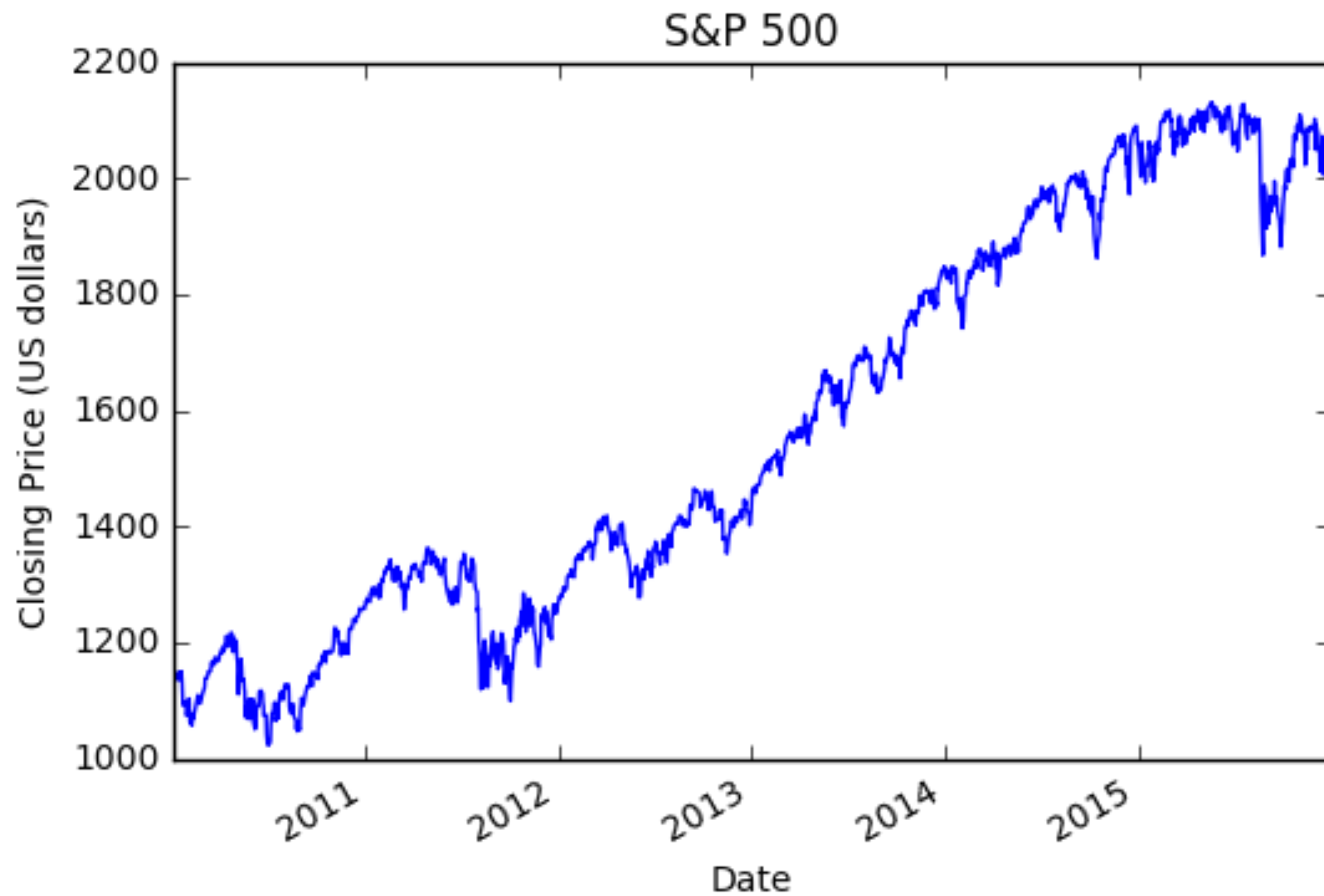
```
In [7]: sp500['Close'].plot(title='S&P 500')
```

```
In [8]: plt.ylabel('Closing Price (US Dollars)')
```

```
In [9]: plt.show()
```



# Labels and title





# One week

```
In [10]: sp500.loc['2012-4-1':'2012-4-7', 'Close'].plot(title='S&P  
....: 500')
```

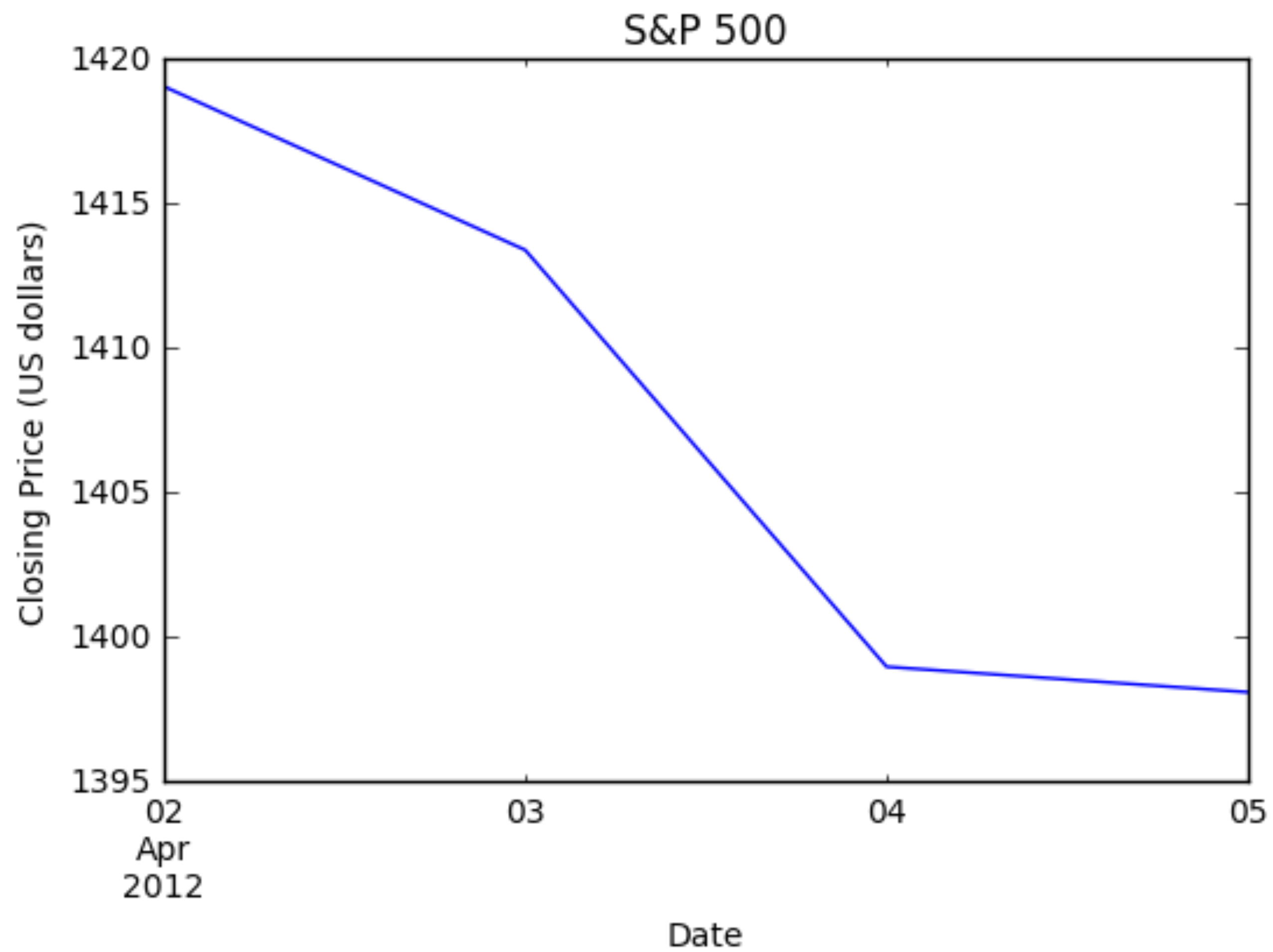
```
In [11]: plt.ylabel('Closing Price (US Dollars)')
```

```
In [12]: plt.show()
```





# One week





# Plot styles

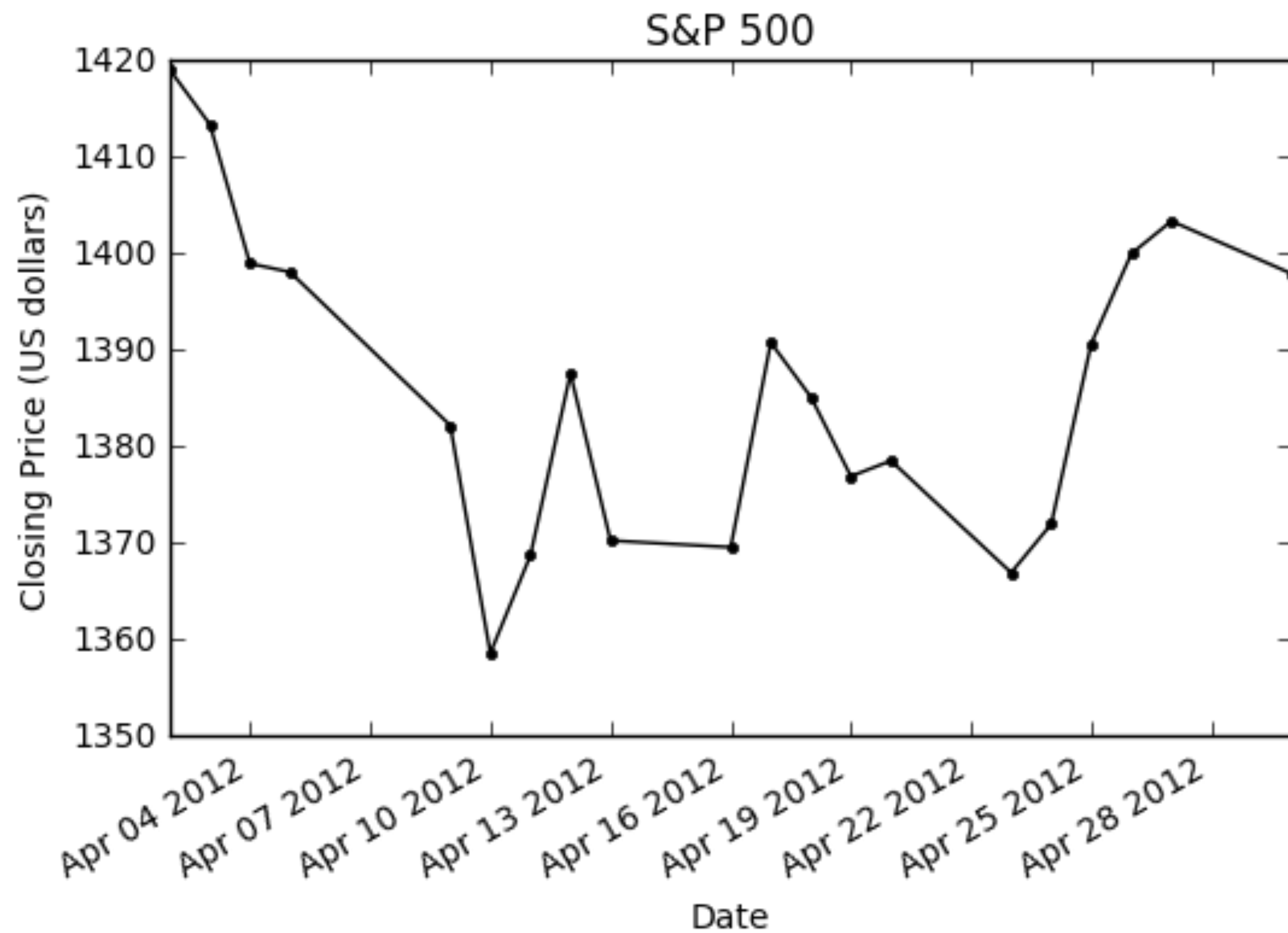
```
In [13]: sp500.loc['2012-4', 'Close'].plot(style='k.-',  
....:                                     title='S&P500')
```

```
In [14]: plt.ylabel('Closing Price (US Dollars)')
```

```
In [15]: plt.show()
```



# One week



# More plot styles

- Style format string
  - color (k: black)
  - marker (. : dot)
  - line type (-: solid)



# More plot styles

Color	Marker	Line
b: blue	o: circle	: dotted
g: green	*: star	–: dashed
r: red	s: square	
c: cyan	+: plus	



# Area plot

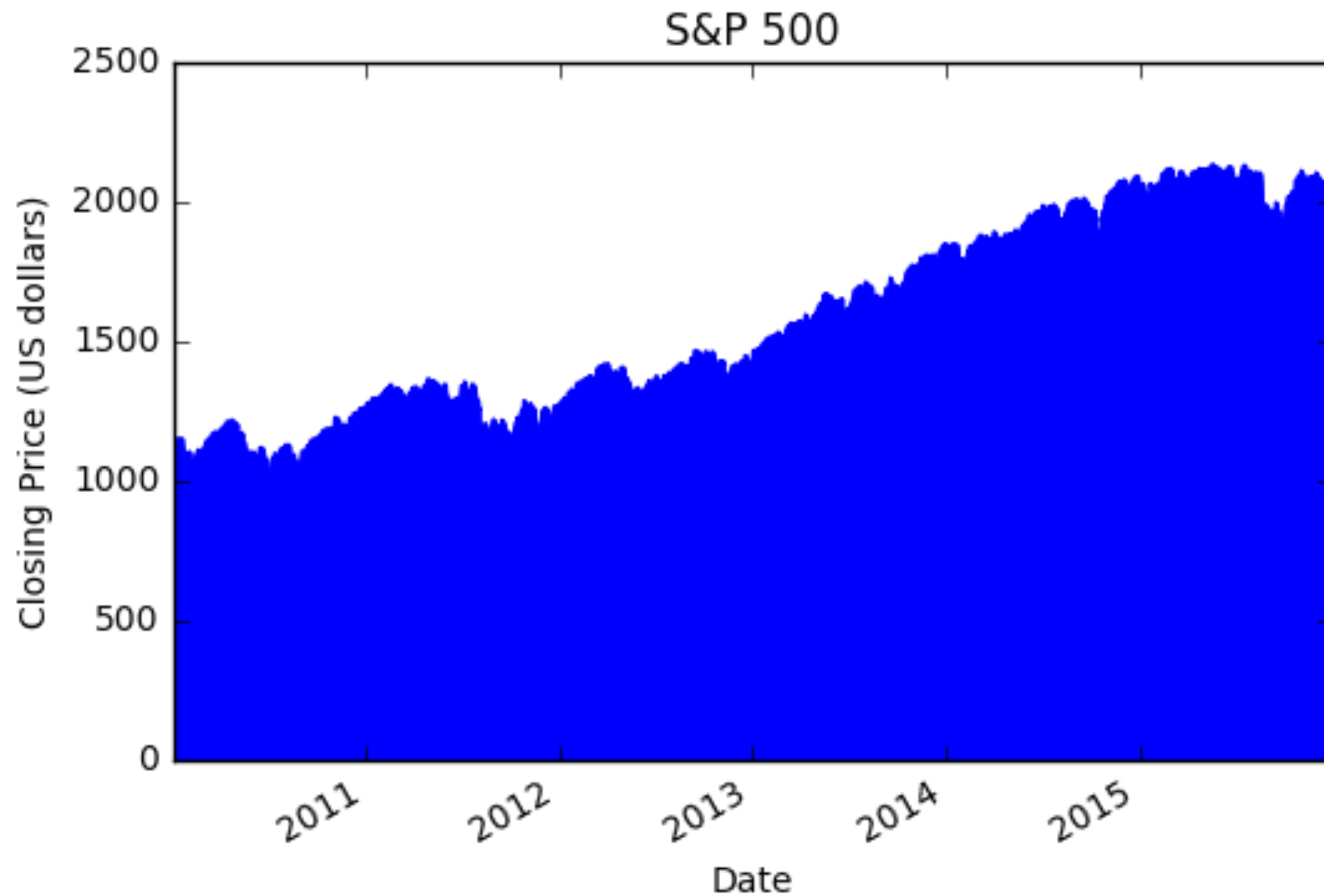
```
In [16]: sp500['Close'].plot(kind='area', title='S&P 500')
```

```
In [17]: plt.ylabel('Closing Price (US Dollars)')
```

```
In [18]: plt.show()
```



# Area plot





# Multiple columns

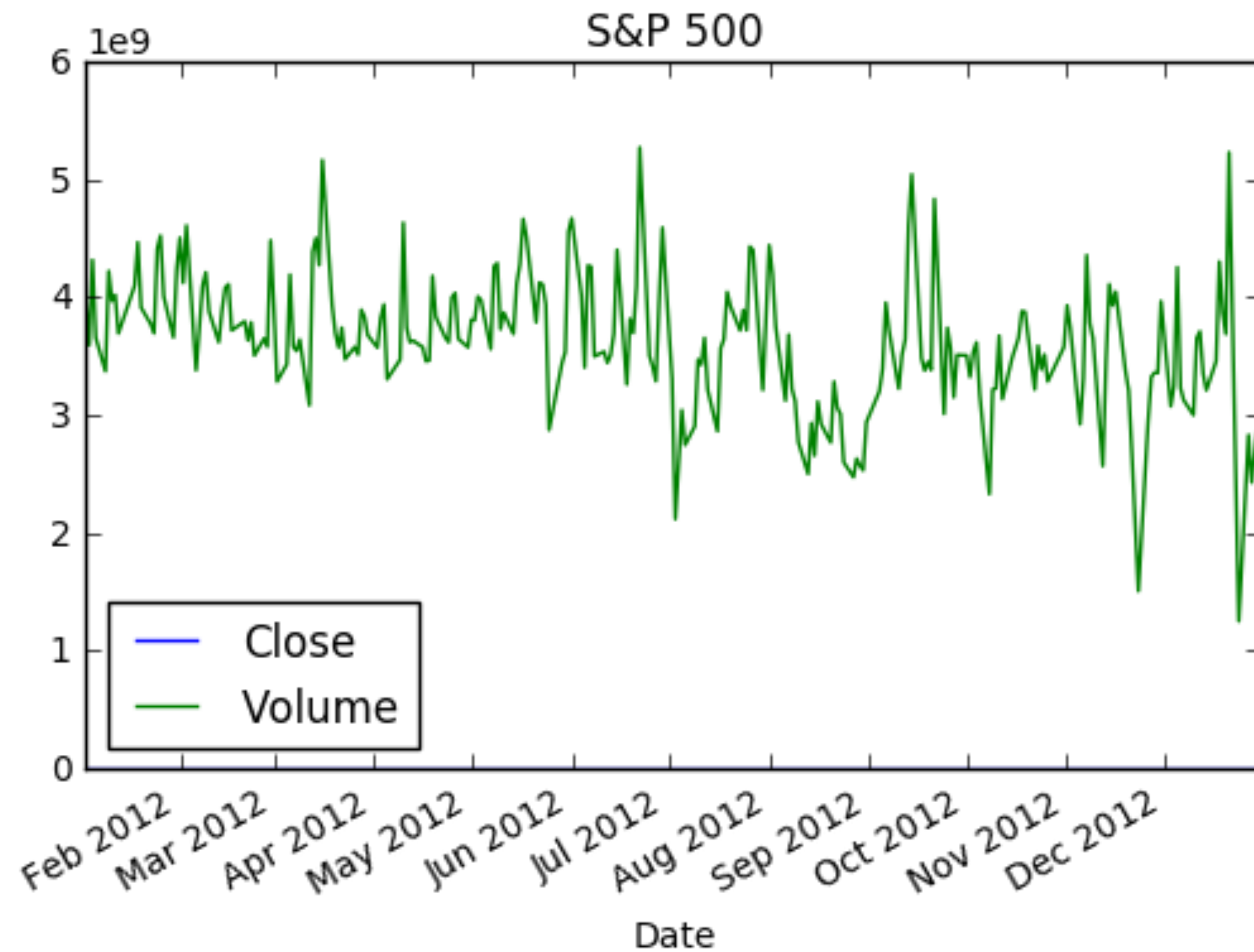
```
In [19]: sp500.loc['2012', ['Close', 'Volume']].plot(title='S&P  
...: 500')
```

```
In [20]: plt.show()
```





# Multiple columns



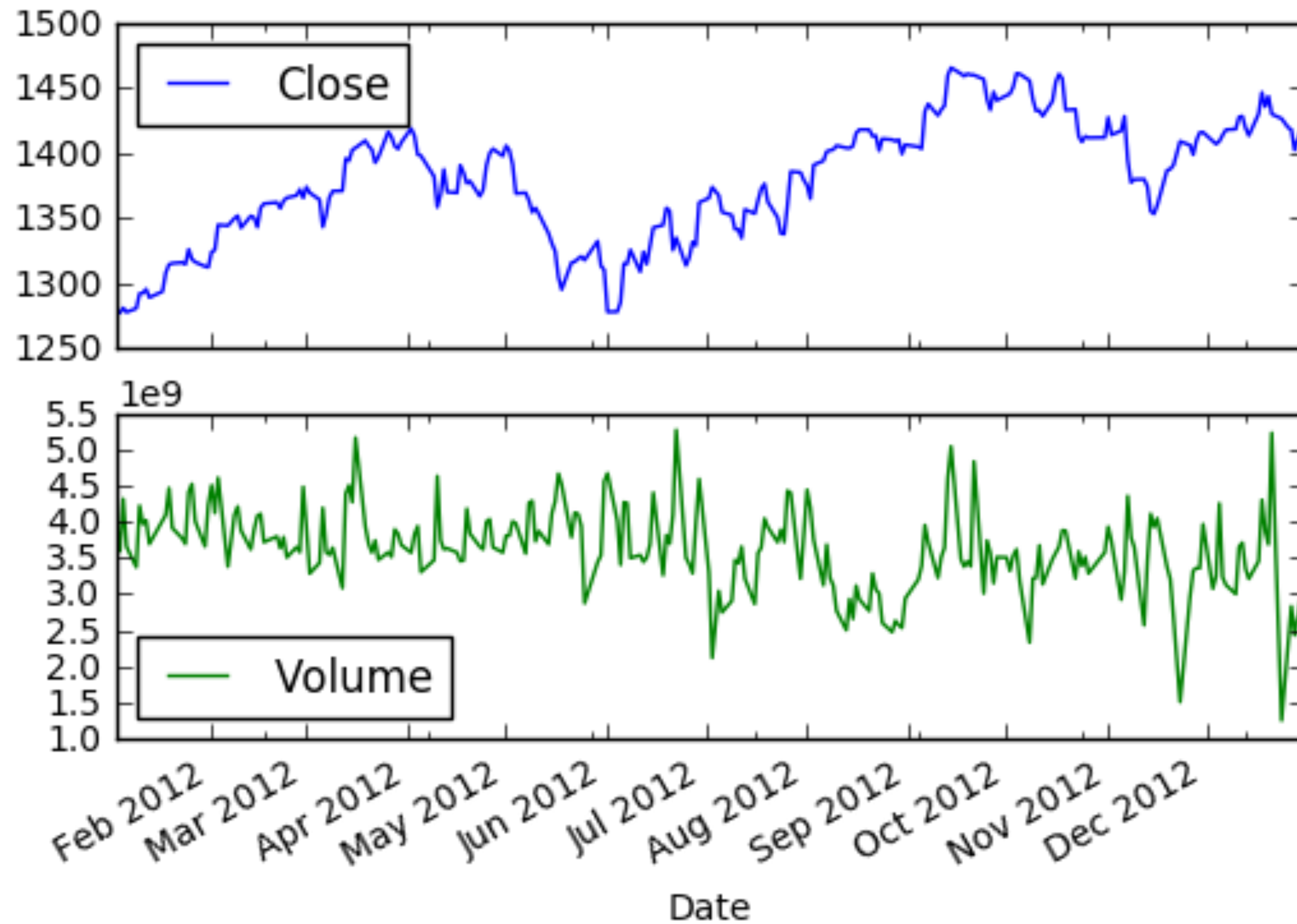


# Subplots

```
In [21]: sp500.loc['2012', ['Close', 'Volume']].plot(subplots=True)  
  
In [22]: plt.show()
```



# Subplots





PANDAS FOUNDATIONS

**Let's practice!**