

Visual exploratory data analysis

PANDAS FOUNDATIONS



Dhavide Aruliah
Director of Training, Anaconda

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

¹ Source: R.A. Fisher, Annual Eugenics, 7, Part II, 179² 188 (1936)
(<http://archive.ics.uci.edu/ml/datasets/Iris>)

Data import

```
import pandas as pd  
import matplotlib.pyplot as plt  
iris = pd.read_csv('iris.csv', index_col=0)  
print(iris.shape)
```

(150, 5)

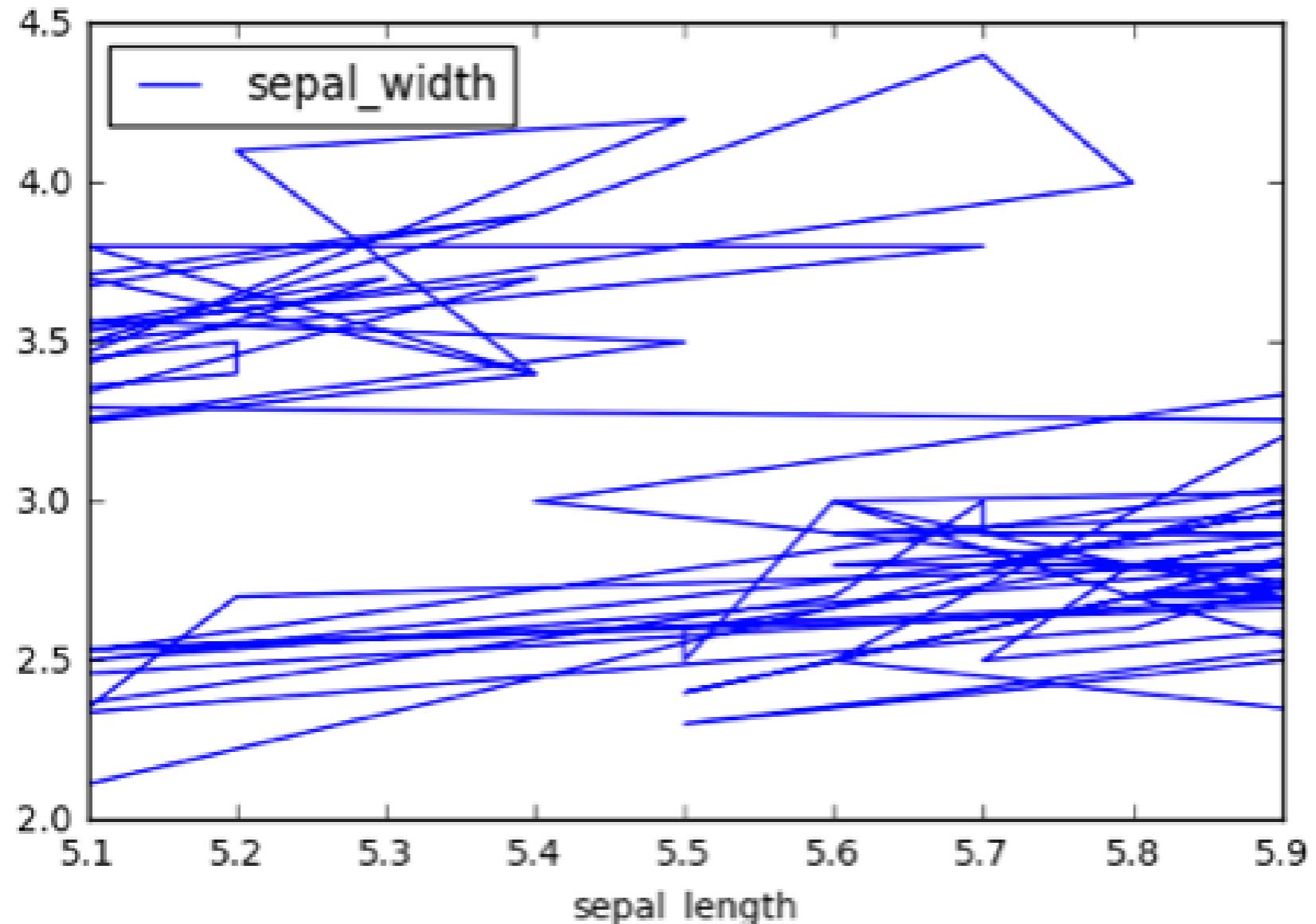
Line plot

```
iris.head()
```

```
  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
```

```
iris.plot(x='sepal_length', y='sepal_width')
plt.show()
```

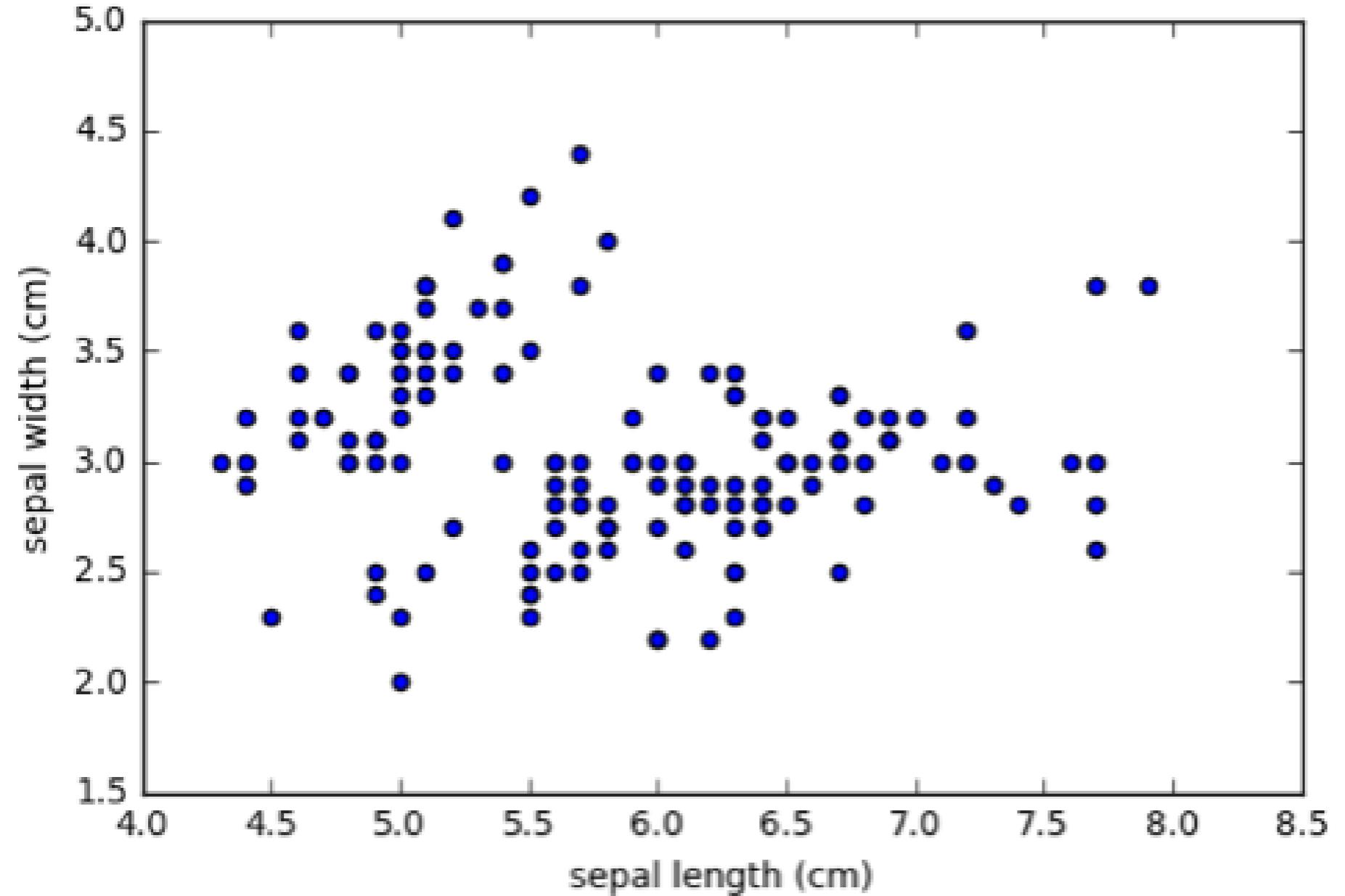
Line plot



Scatter plot

```
iris.plot(x='sepal_length', y='sepal_width',  
          kind='scatter')  
plt.xlabel('sepal length (cm)')  
plt.ylabel('sepal width (cm)')  
plt.show()
```

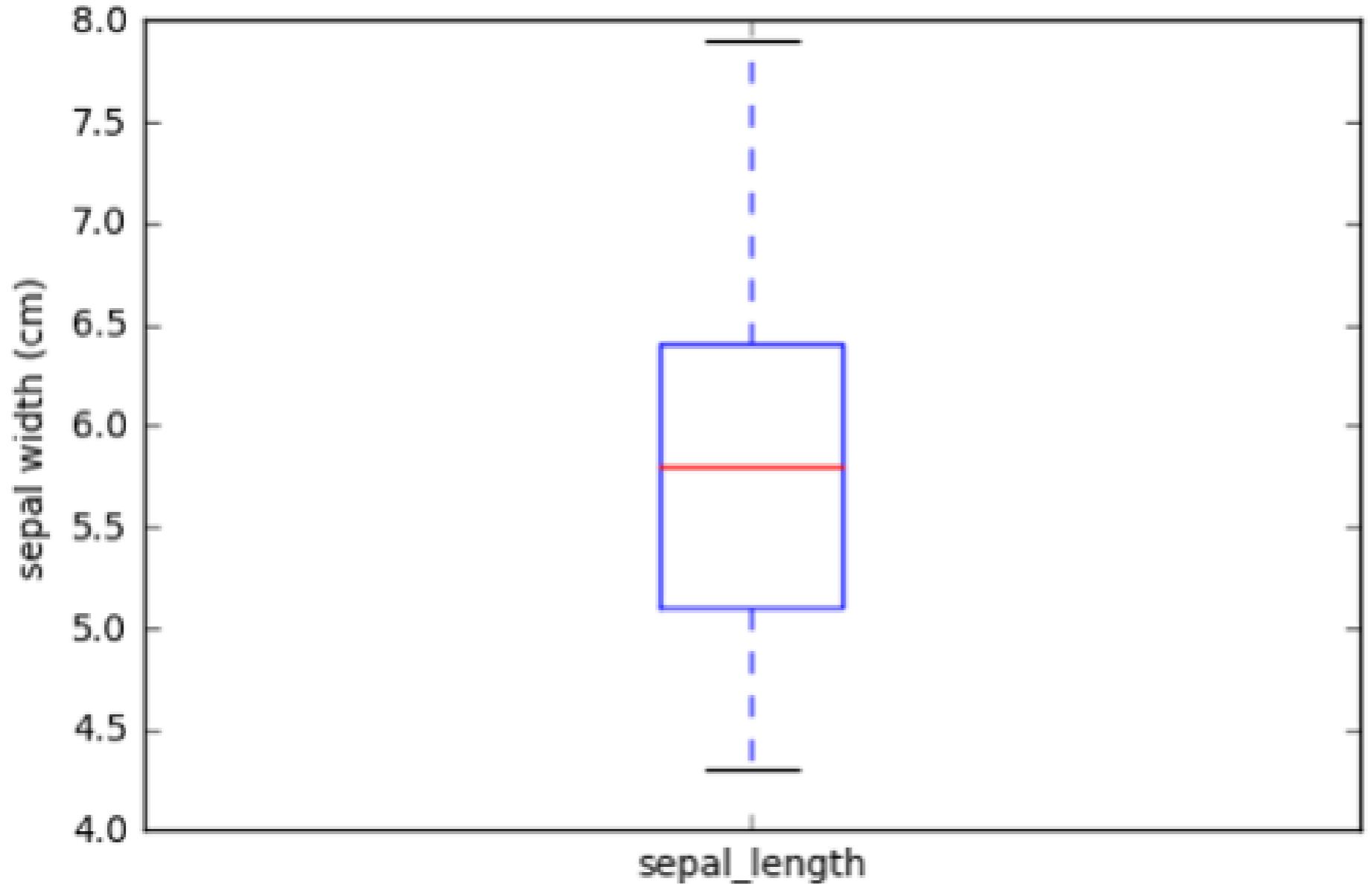
Scatter plot



Box plot

```
iris.plot(y='sepal_length', kind='box')
plt.ylabel('sepal width (cm)')
plt.show()
```

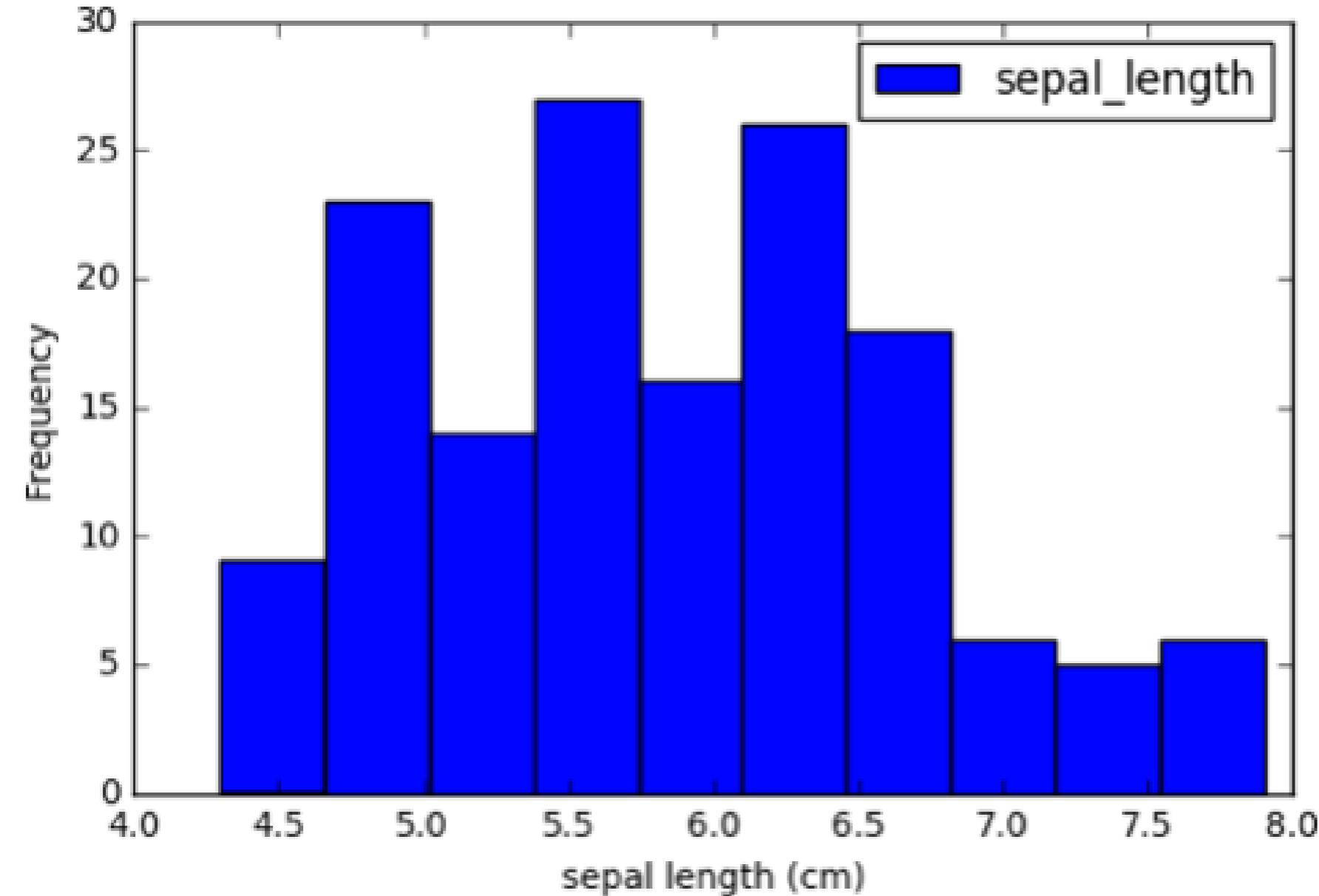
Box plot



Histogram

```
iris.plot(y='sepal_length', kind='hist')  
plt.xlabel('sepal length (cm)')  
plt.show()
```

Histogram



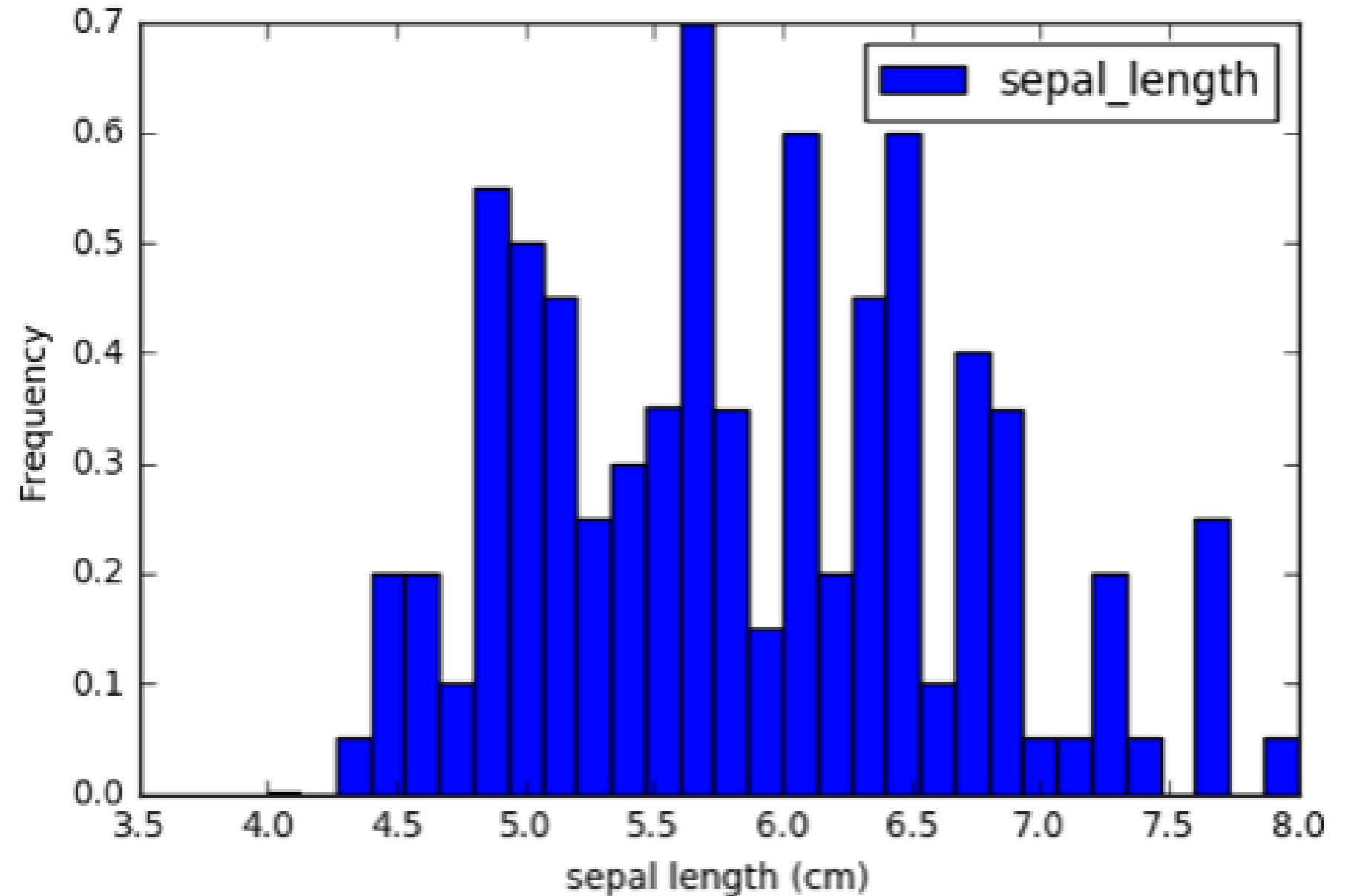
Histogram options

- 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

```
iris.plot(y='sepal_length', kind='hist',  
          bins=30, range=(4, 8), normed=True)  
plt.xlabel('sepal length (cm)')  
plt.show()
```

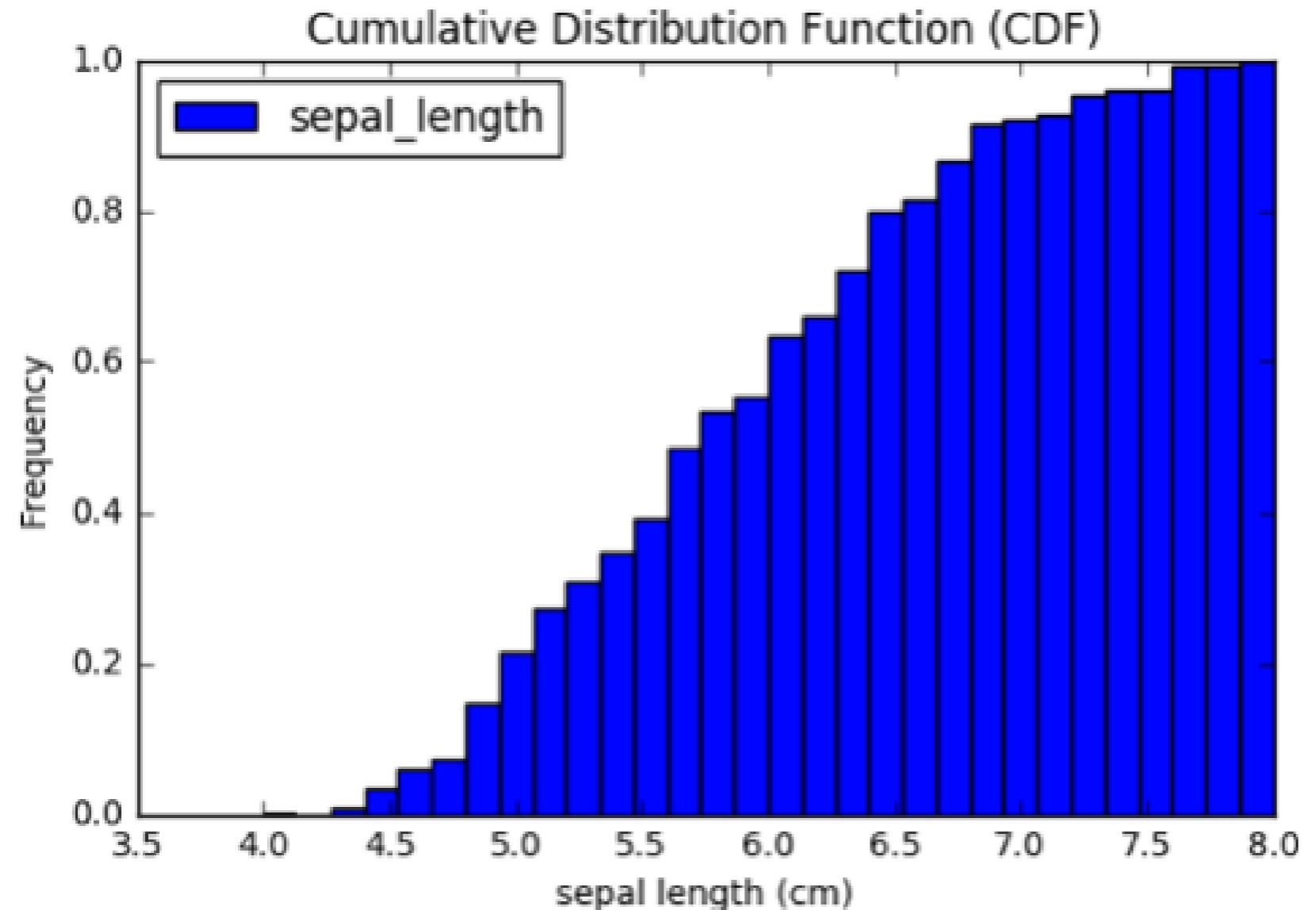
Customizing histogram



Cumulative distribution

```
iris.plot(y='sepal_length', kind='hist', bins=30,  
          range=(4, 8), cumulative=True, normed=True)  
plt.xlabel('sepal length (cm)')  
plt.title('Cumulative distribution function (CDF)')  
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!

Let's practice!

PANDAS FOUNDATIONS

Statistical exploratory data analysis

PANDAS FOUNDATIONS



Dhavide Aruliah
Director of Training, Anaconda

Summarizing with describe()

```
iris.describe() # summary statistics
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.00000	150.00000	150.00000	150.00000
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

```
iris['sepal_length'].count() # Applied to Series
```

```
150
```

```
iris['sepal_width'].count() # Applied to Series
```

```
150
```

```
iris[['petal_length', 'petal_width']].count() # Applied to DataFrame
```

```
petal_length    150
petal_width     150
dtype: int64
```

```
type(iris[['petal_length', 'petal_width']].count()) # returns Series
```

```
pandas.core.series.Series
```

Averages

```
iris['sepal_length'].mean() # Applied to Series
```

```
5.843333333333335
```

```
iris.mean() # Applied to entire DataFrame
```

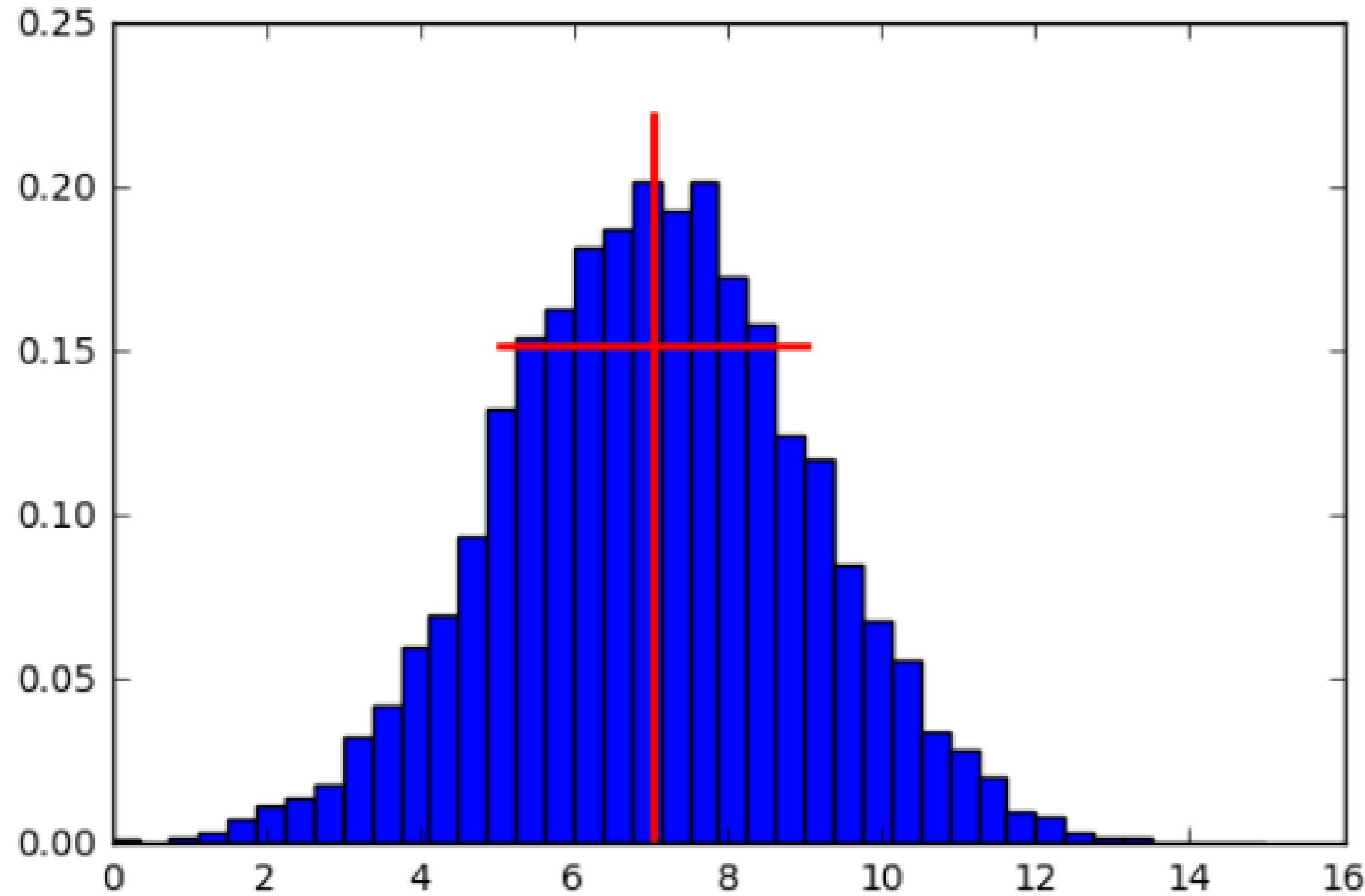
```
sepal_length      5.84333  
sepal_width       3.05733  
petal_length      3.75800  
petal_width       1.19933  
dtype: float64
```

Standard deviations

```
iris.std()
```

```
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

```
iris.median()
```

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

Medians & 0.5 quantiles

```
iris.median()
```

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

```
q = 0.5
iris.quantile(q)
```

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

Inter-quartile range (IQR)

```
q = [0.25, 0.75]  
iris.quantile(q)
```

```
    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

```
iris.min()
```

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

```
iris.max()
```

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

Box plots

```
iris.plot(kind= 'box')
```

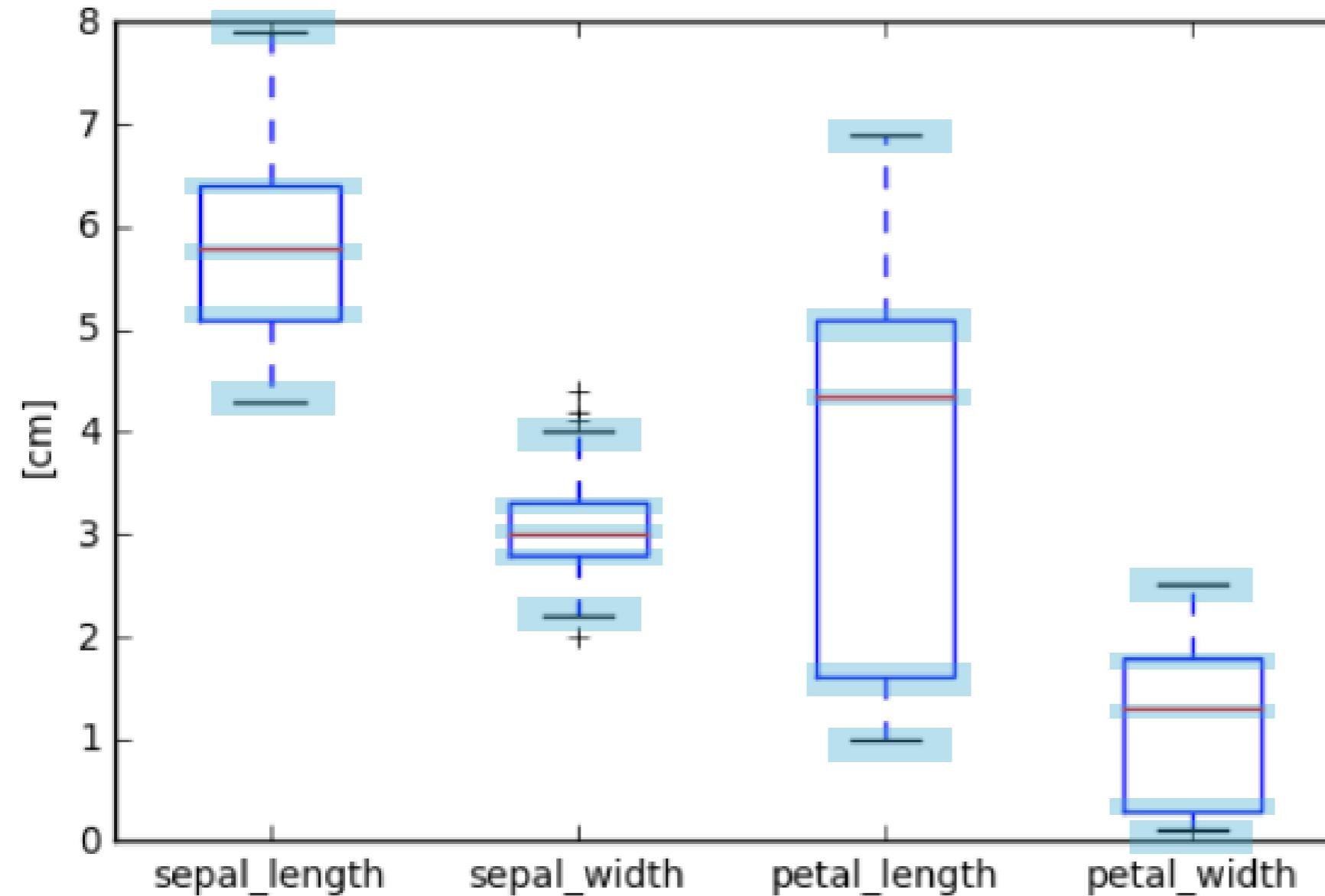
```
<matplotlib.axes._subplots.AxesSubplot at 0x118a3d5f8>
```

```
plt.ylabel(' [cm] ')
```

```
<matplotlib.text.Text at 0x118a524e0>
```

```
plt.show()
```

Box plots



Percentiles as quantiles

```
iris.describe() # summary statistics
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.00000	150.00000	150.00000	150.00000
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

Let's practice!

PANDAS FOUNDATIONS

Separating populations

PANDAS FOUNDATIONS



Dhavide Aruliah
Director of Training, Anaconda

```
iris.head()
```

```
  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

```
iris['species'].describe()
```

```
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

```
iris['species'].unique()
```

```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

Filtering by species

```
indices = iris['species'] == 'setosa'  
setosa = iris.loc[indices, :] # extract new DataFrame  
  
indices = iris['species'] == 'versicolor'  
versicolor = iris.loc[indices, :] # extract new DataFrame  
  
indices = iris['species'] == 'virginica'  
virginica = iris.loc[indices, :] # extract new DataFrame
```

Checking species

```
setosa['species'].unique()
```

```
array(['setosa'], dtype=object)
```

```
versicolor['species'].unique()
```

```
array(['versicolor'], dtype=object)
```

```
virginica['species'].unique()
```

```
array(['virginica'], dtype=object)
```

```
del setosa['species'], versicolor['species'],  
virginica['species']
```

Checking indexes

```
setosa.head(2)
```

```
    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
```

```
versicolor.head(2)
```

```
    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
```

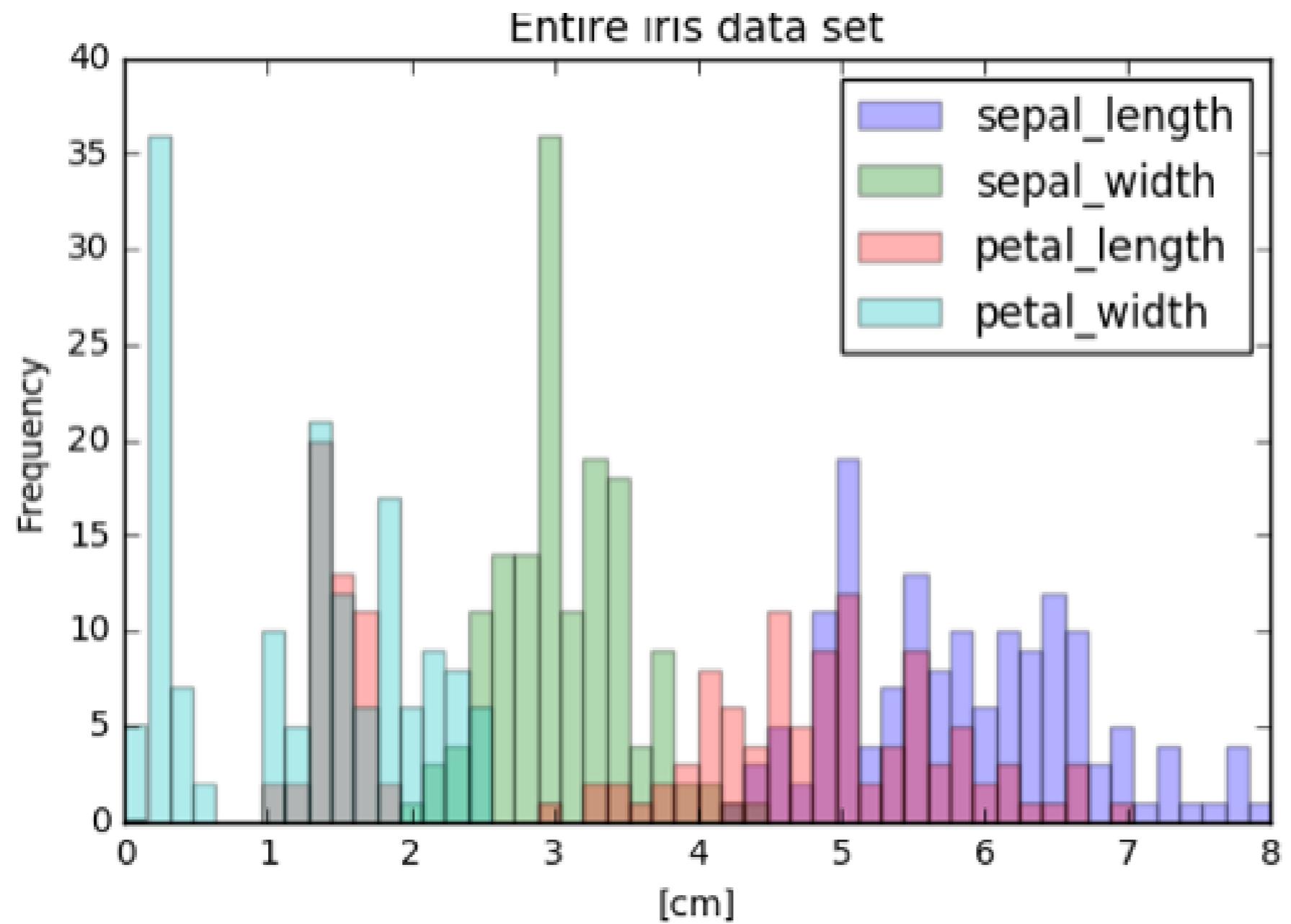
```
versicolor.head(2)
```

```
    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

```
iris.plot(kind= 'hist', bins=50, range=(0,8), alpha=0.3)
plt.title('Entire iris data set')
plt.xlabel(' [cm]')
plt.show()
```

Visual EDA: all data



Visual EDA: individual factors

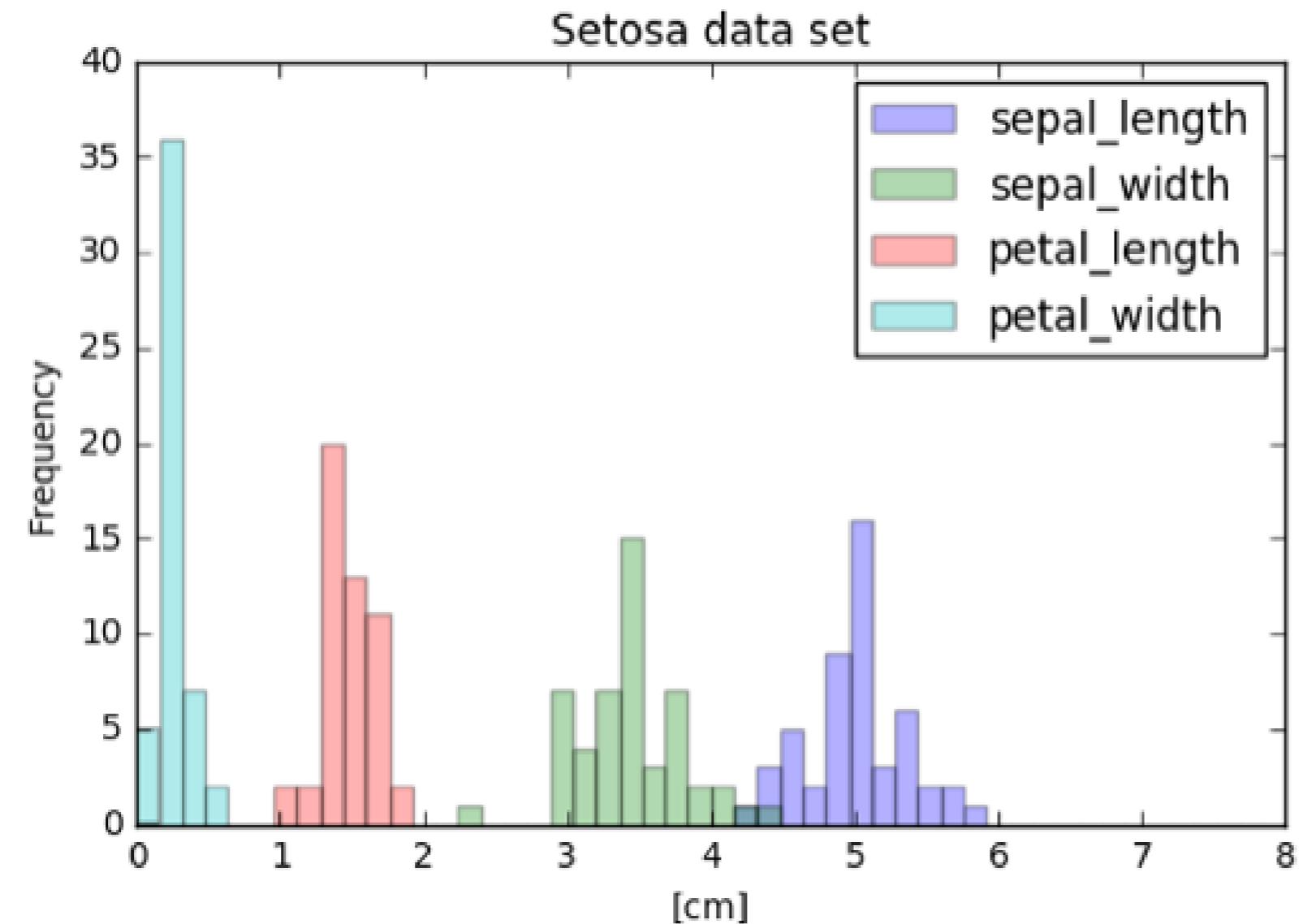
```
setosa.plot(kind='hist', bins=50, range=(0, 8), alpha=0.3)
plt.title('Setosa data set')
plt.xlabel('[cm]')

versicolor.plot(kind='hist', bins=50, range=(0, 8), alpha=0.3)
plt.title('Versicolor data set')
plt.xlabel('[cm]')

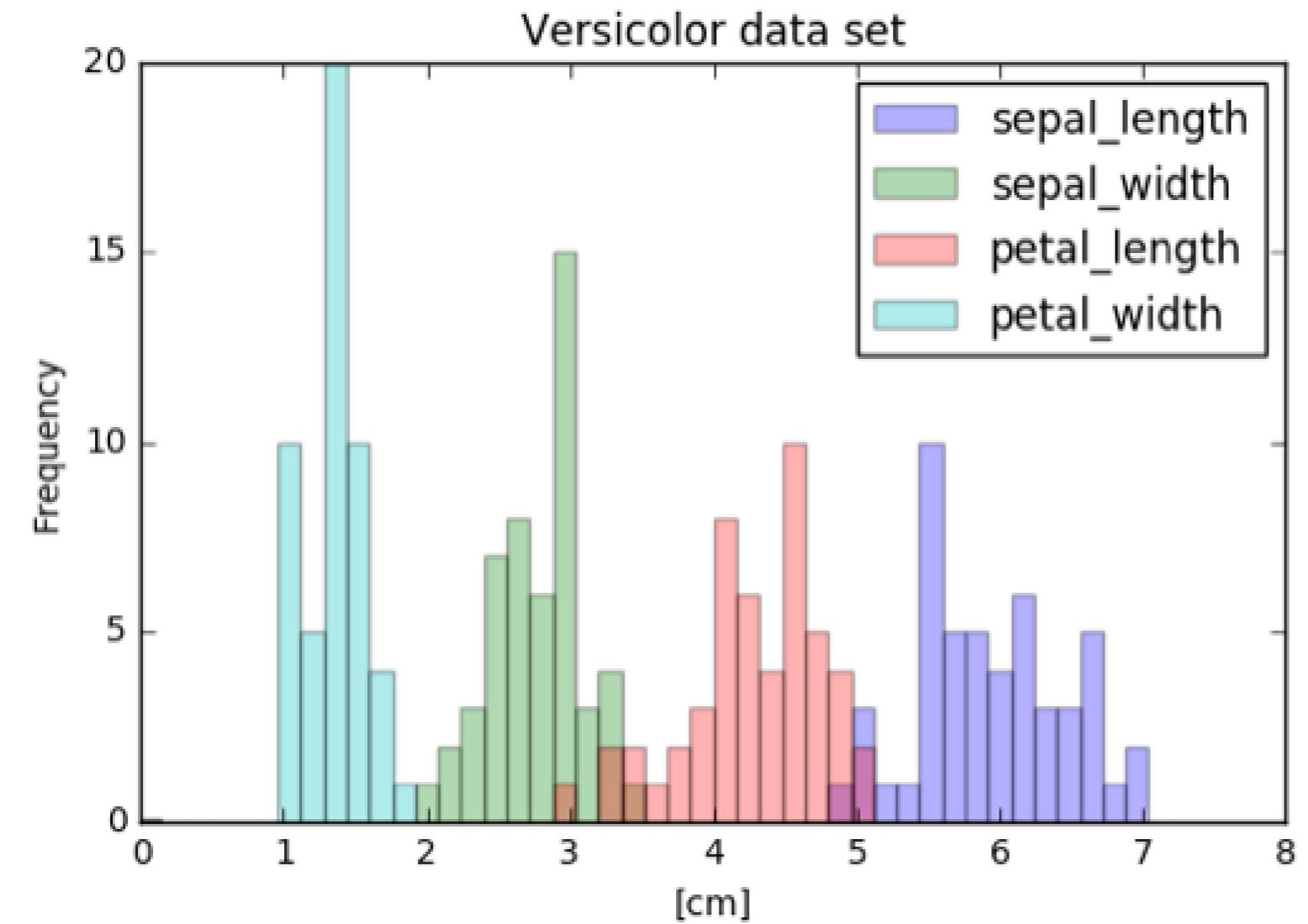
virginica.plot(kind='hist', bins=50, range=(0, 8), alpha=0.3)
plt.title('Virginica data set')
plt.xlabel('[cm]')

plt.show()
```

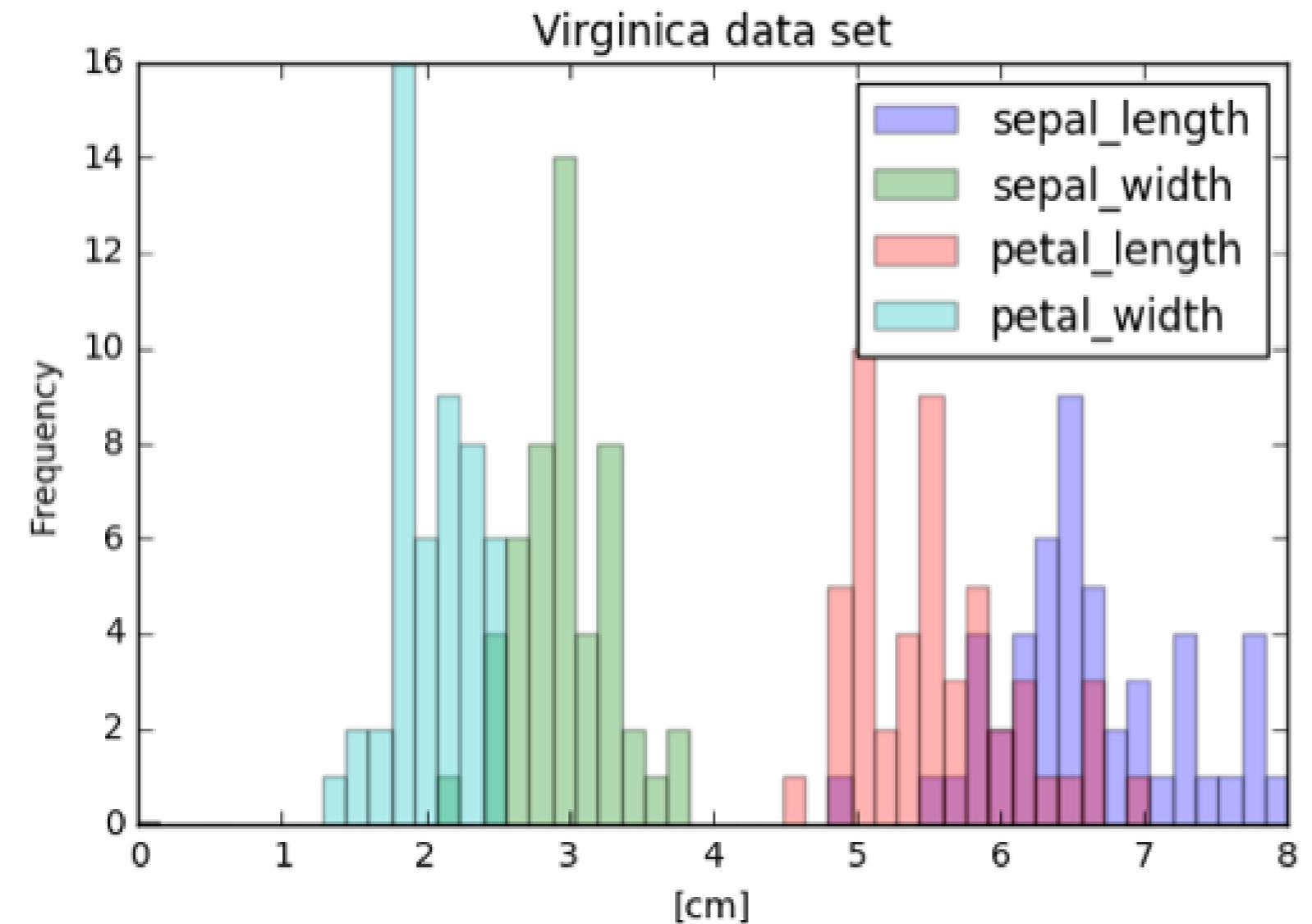
Visual EDA: Setosa data



Visual EDA: Versicolor data



Visual EDA: Virginica data



Statistical EDA: describe()

```
describe_all = iris.describe()  
print(describe_all)
```

	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_setosa = setosa.describe()  
describe_versicolor = versicolor.describe()  
describe_virginica = virginica.describe()
```

Computing errors

```
error_setosa = 100 * np.abs(describe_setosa - describe_all)
error_setosa = error_setosa/describe_setosa

error_versicolor = 100 * np.abs(describe_versicolor - describe_all)
error_versicolor = error_versicolor/describe_versicolor

error_virginica = 100 * np.abs(describe_virginica - describe_all)
error_virginica = error_verginica/describe_virginica
```

Viewing errors

```
print(error_setosa)
```

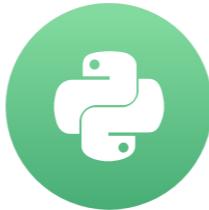
	sepal_length	sepal_width	petal_length	petal_width
count	200.000000	200.000000	200.000000	200.000000
mean	16.726595	10.812913	157.045144	387.533875
std	134.919250	14.984768	916.502136	623.284534
min	0.000000	13.043478	0.000000	0.000000
25%	6.250000	12.500000	14.285714	50.000000
50%	16.000000	11.764706	190.000000	550.000000
75%	23.076923	10.204082	223.809524	500.000000
max	36.206897	0.000000	263.157895	316.666667

Let's practice!

PANDAS FOUNDATIONS

Indexing time series

PANDAS FOUNDATIONS



Dhavide Aruliah

Director of Training, Anaconda

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

```
import pandas as pd  
sales = pd.read_csv('sales-feb-2015.csv',  
                     parse_dates=True, index_col='Date')
```

Parse dates

```
sales.head()
```

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

```
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

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

```
'Mediacore'
```

Selecting whole day

```
sales.loc['2015-2-5']
```

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

```
sales.loc['2015-2']
```

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

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

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

```
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'])  
evening_2_11
```

```
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

```
sales.reindex(evening_2_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

```
sales.reindex(evening_2_11, method='ffill')
```

```
    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
```

```
sales.reindex(evening_2_11, method='bfill')
```

```
    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
```

Let's practice!

PANDAS FOUNDATIONS

Resampling time series data

PANDAS FOUNDATIONS



Dhavide Aruliah
Director of Training, Anaconda

Sales data

```
import pandas as pd  
sales = pd.read_csv('sales-feb-2015.csv',  
                     parse_dates=True, index_col='Date')  
sales.head()
```

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

Resampling

- Statistical methods over different time intervals
 - mean(), sum(), count(), etc.
- Downsampling
 - reduce datetime rows to slower frequency
- Upsampling
 - increase datetime rows to faster frequency

Aggregating means

```
daily_mean = sales.resample('D').mean()  
daily_mean
```

```
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

```
print(daily_mean.loc['2015-2-2'])
```

```
Units      6.0
Name: 2015-02-02 00:00:00, dtype: float64
```

```
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
```

```
sales.loc['2015-2-2', 'Units'].mean()
```

```
6.0
```

Method chaining

```
sales.resample('D').sum()
```

```
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

```
sales.resample('D').sum().max()
```

```
Units      29.0
```

```
dtype: float64
```

Resampling strings

```
sales.resample('W').count()
```

Date	Company	Product	Units
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

```
sales.loc[:, 'Units'].resample('2W').sum()
```

```
Date
2015-02-08    82
2015-02-22    79
2015-03-08    14
Freq: 2W-SUN, Name: Units, dtype: int64
```

Upsampling

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

```
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

```
two_days.resample('4H').ffill()
```

```
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
```

Let's practice!

PANDAS FOUNDATIONS

Manipulating time series data

PANDAS FOUNDATIONS



Dhavide Aruliah
Director of Training, Anaconda

Sales data

```
import pandas as pd  
sales = pd.read_csv('sales-feb-2015.csv',  
                     parse_dates=[ 'Date' ])  
sales.head()
```

	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

```
sales['Company'].str.upper()
```

```
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

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

```
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

True + False

1

True + True

2

False + False

0

Boolean reduction

```
sales['Product'].str.contains('ware').sum()
```

14

Datetime methods

```
sales[ 'Date' ].dt.hour
```

```
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

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

```
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

```
central.dt.tz_convert('US/Eastern')
```

```
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

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

```
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

```
population = pd.read_csv('world_population.csv',  
                         parse_dates=True, index_col='Date')  
population
```

```
Population  
Date  
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

```
population.resample('A').first()
```

```
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

```
population.resample('A').first().interpolate('linear')
```

```
Population  
Date  
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
```

Let's practice!

PANDAS FOUNDATIONS

Time series visualization

PANDAS FOUNDATIONS



Dhavide Aruliah
Director of Training, Anaconda

Topics

- Line types
- Plot types
- Subplots

S&P 500 Data

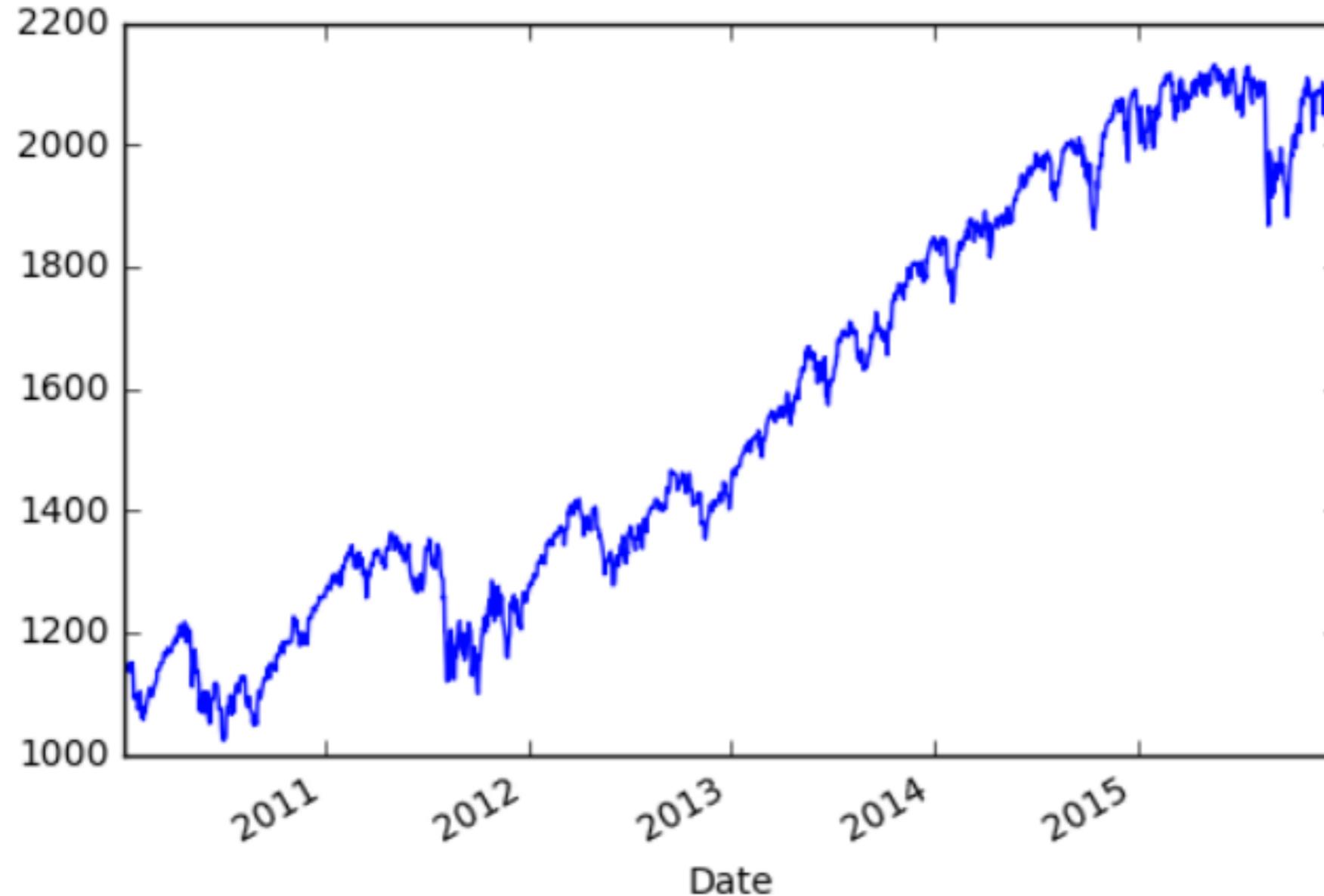
```
import pandas as pd
import matplotlib.pyplot as plt
sp500 = pd.read_csv('sp500.csv', parse_dates=True,
                    index_col= 'Date')
sp500.head()
```

	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

```
sp500['Close'].plot()  
plt.show()
```

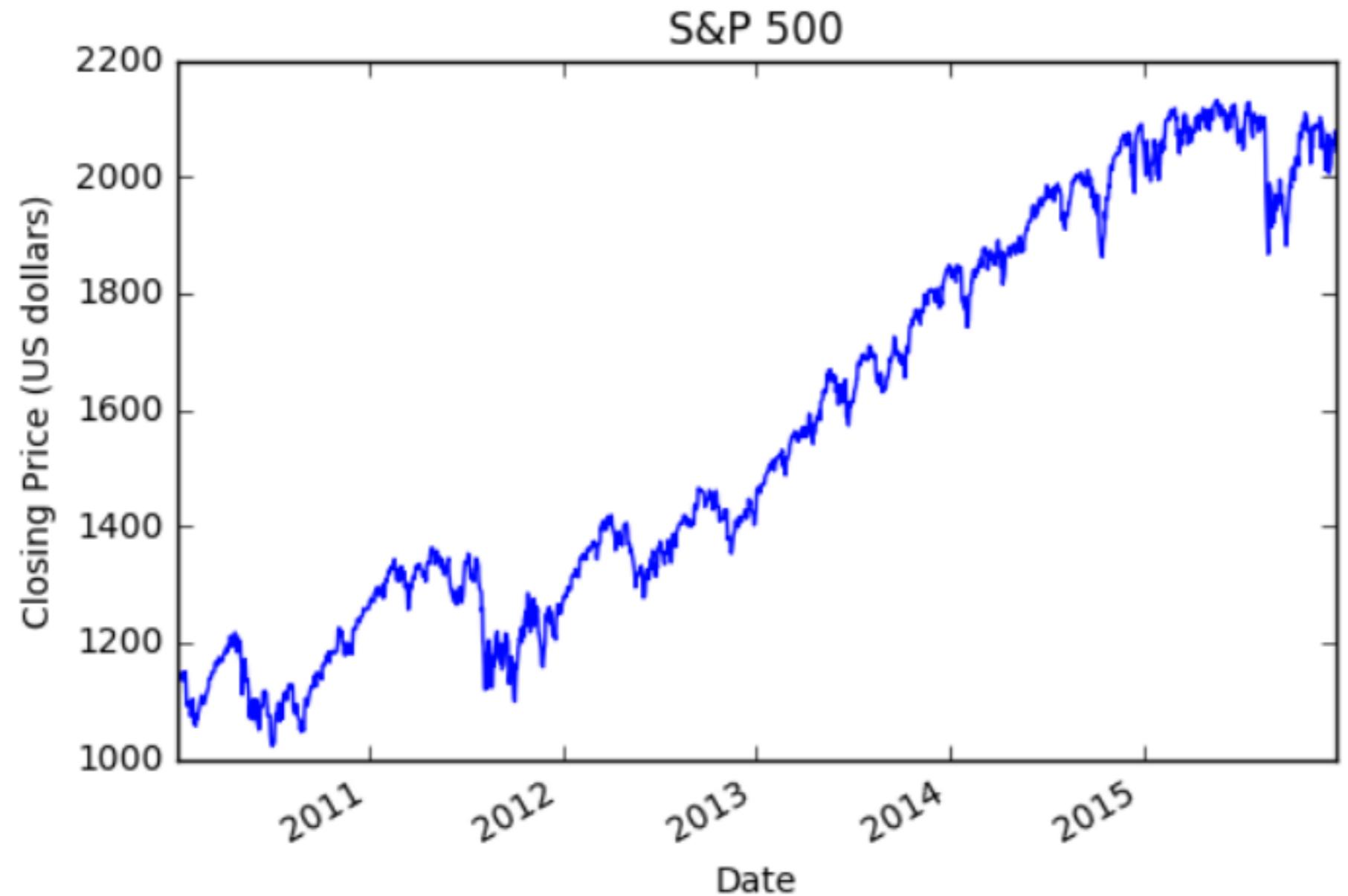
Default plot



Labels and title

```
sp500['Close'].plot(title='S&P 500')
plt.ylabel('Closing Price (US Dollars)')
plt.show()
```

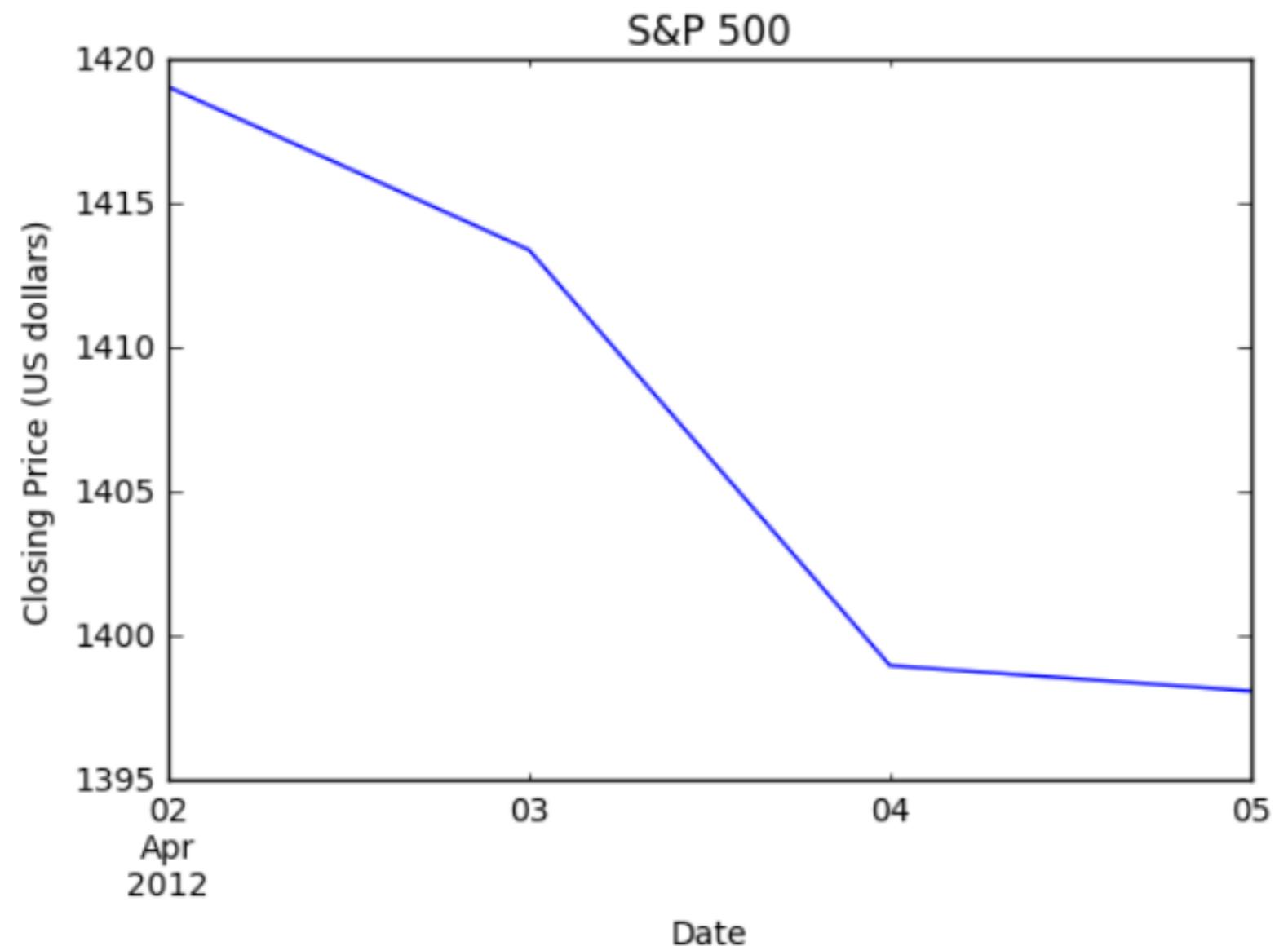
Labels and title



One week

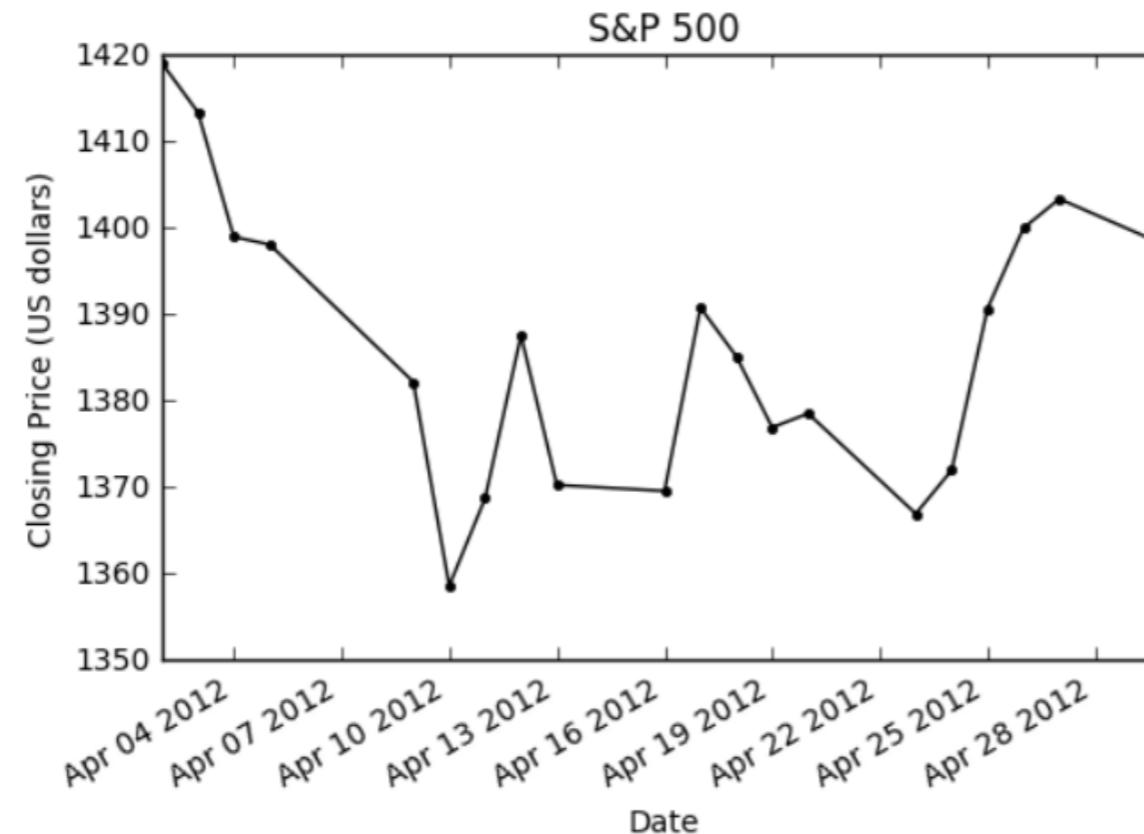
```
sp500.loc['2012-4-1':'2012-4-7', 'Close'].plot(title='S&P  
500')  
plt.ylabel('Closing Price (US Dollars)')  
plt.show()
```

One week



Plot styles

```
sp500.loc['2012-4', 'Close'].plot(style='k.-',  
                                    title='S&P500')  
plt.ylabel('Closing Price (US Dollars)')  
plt.show()
```



More plot styles

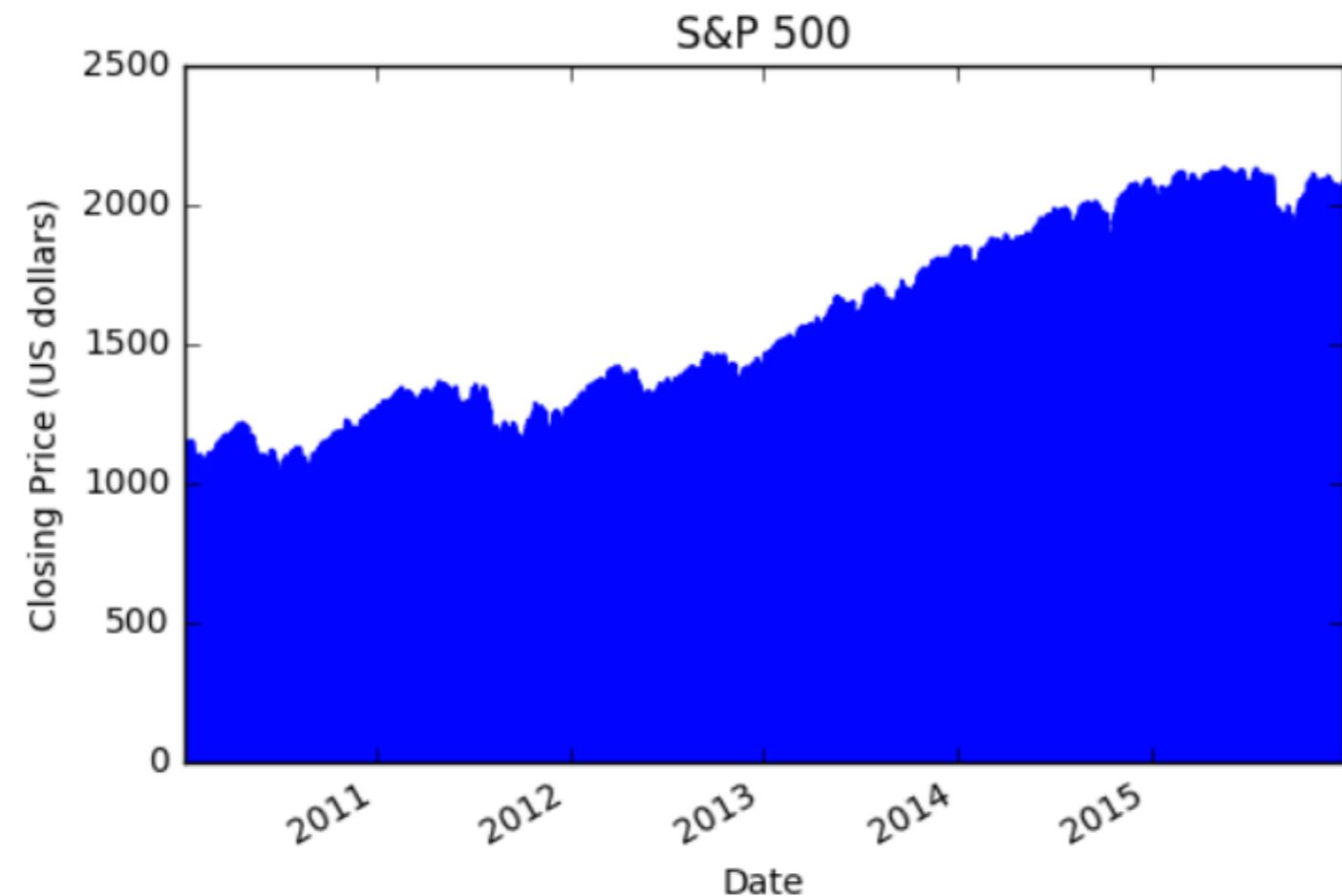
- Style format string
 - color (k: black)
 - marker (.: dot)
 - line type (-: solid)

More plot styles

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

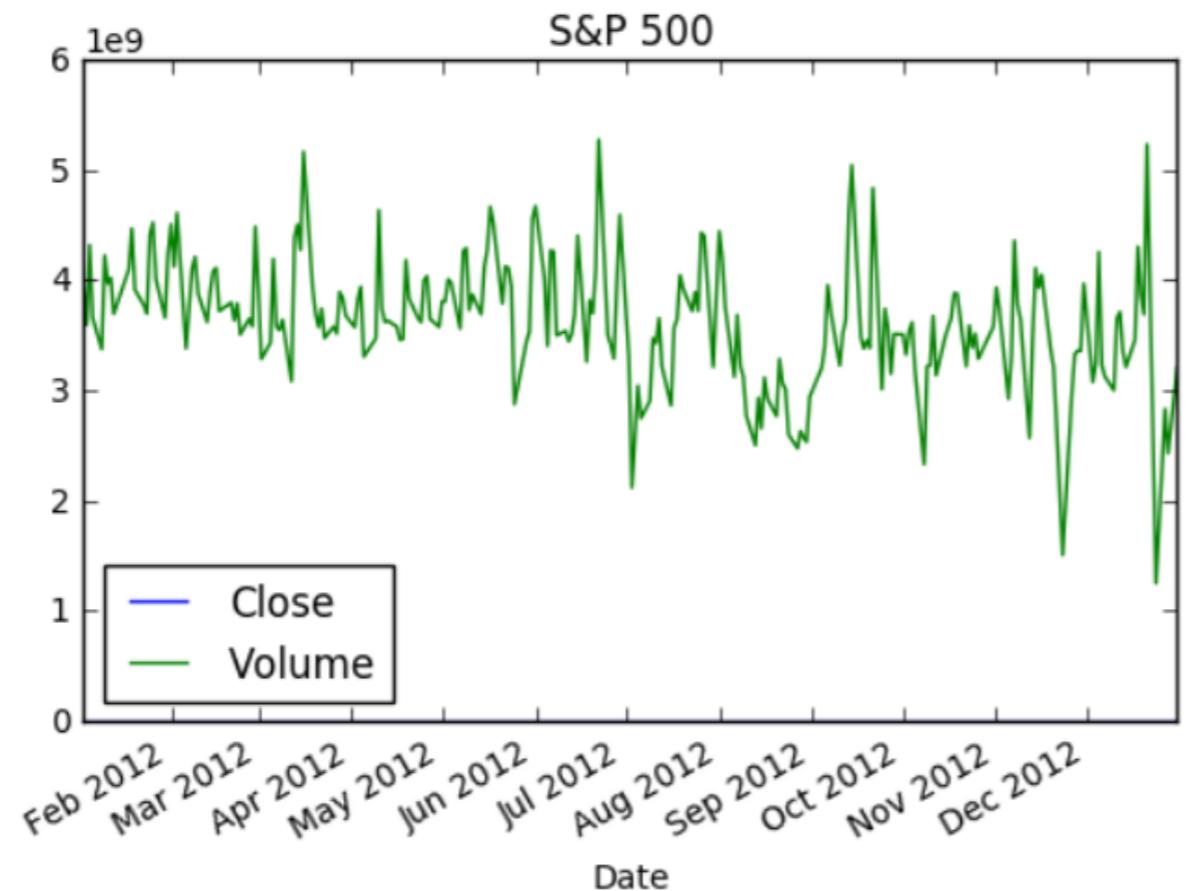
Area plot

```
sp500['Close'].plot(kind='area', title='S&P 500')
plt.ylabel('Closing Price (US Dollars)')
plt.show()
```



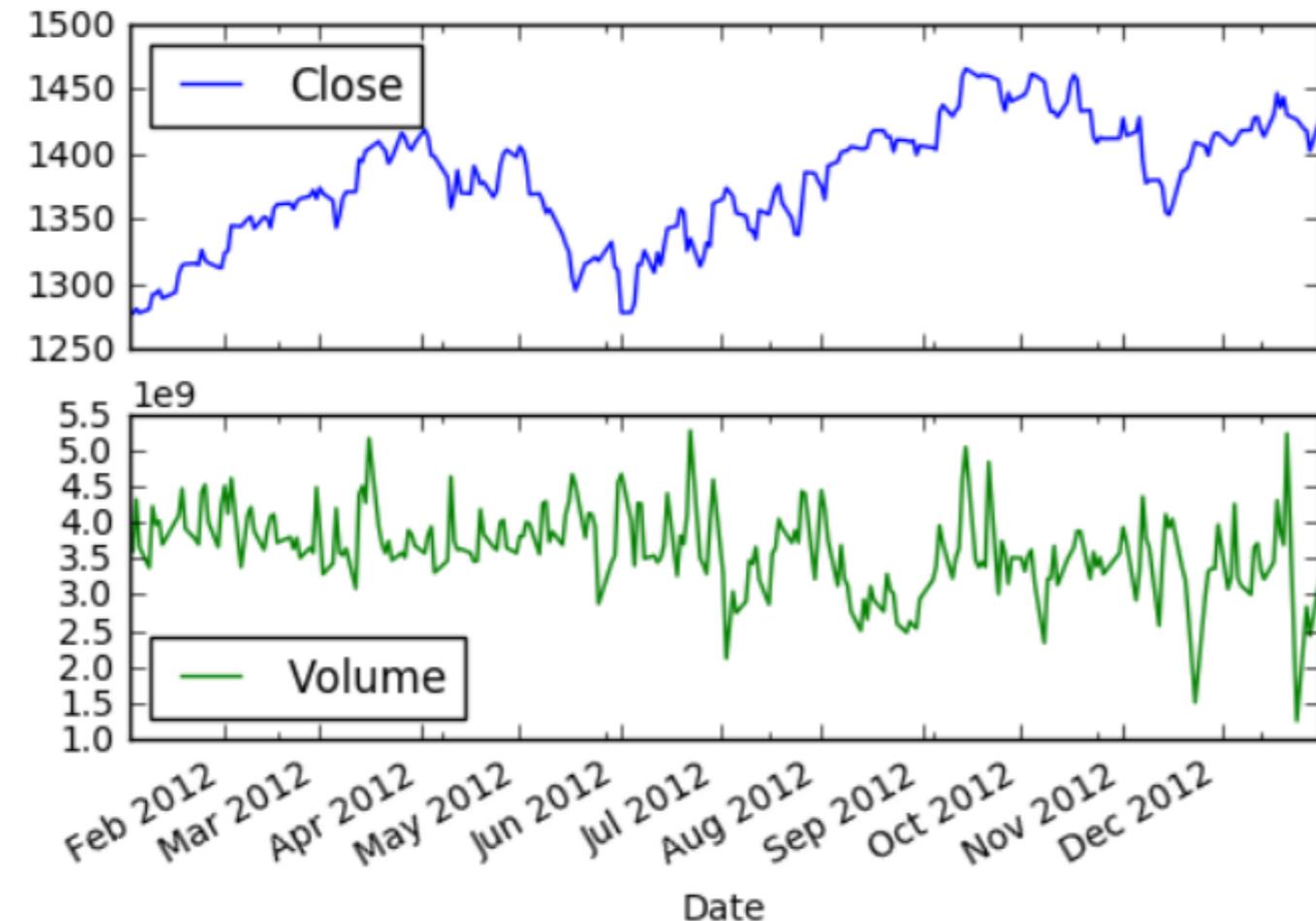
Multiple columns

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



Subplots

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



Let's practice!

PANDAS FOUNDATIONS

Reading and cleaning the data

PANDAS FOUNDATIONS



Dhavide Aruliah
Director of Training, Anaconda

Case study

- Comparing observed weather data from two sources

	Temperature	DewPoint	Pressure	Date
0	46.2	37.5	1.0	20100101 00:00
1	44.6	37.1	1.0	20100101 01:00
2	44.1	36.9	1.0	20100101 02:00
3	43.8	36.9	1.0	20100101 03:00
4	43.5	36.8	1.0	20100101 04:00

	Date	Wban	...	station_pressure	sea_level_pressure
0	2011-01-01 00:53:00	13904	...	29.42	29.95
1	2011-01-01 01:53:00	13904	...	29.49	30.01
2	2011-01-01 02:53:00	13904	...	29.49	30.01
3	2011-01-01 03:53:00	13904	...	29.51	30.03
4	2011-01-01 04:53:00	13904	...	29.51	30.04

¹ Source: National Oceanic & Atmospheric Administration
(www.noaa.gov/climate)

Climate normals of Austin, TX from 1981-2010

	Temperature	DewPoint	Pressure	Date
0	46.2	37.5	1.0	20100101 00:00
1	44.6	37.1	1.0	20100101 01:00
2	44.1	36.9	1.0	20100101 02:00
3	43.8	36.9	1.0	20100101 03:00
4	43.5	36.8	1.0	20100101 04:00
5	43.0	36.5	1.0	20100101 05:00
6	43.1	36.3	1.0	20100101 06:00
7	42.3	35.9	1.0	20100101 07:00
8	42.5	36.2	1.0	20100101 08:00
9	45.9	37.8	1.0	20100101 09:00

Source: National Oceanic & Atmospheric Administration,
www.noaa.gov/climate

Weather data of Austin, TX from 2011

	Date	Wban	date	Time	StationType	...	relative_humidity	wind_speed	wind_direction	station_pressure	sea_level_pressure
0	2011-01-01 00:53:00	13904	20110101	5300	12	...	24.0	15.0	360	29.42	29.95
1	2011-01-01 01:53:00	13904	20110101	15300	12	...	23.0	10.0	340	29.49	30.01
2	2011-01-01 02:53:00	13904	20110101	25300	12	...	22.0	15.0	010	29.49	30.01
3	2011-01-01 03:53:00	13904	20110101	35300	12	...	27.0	7.0	350	29.51	30.03
4	2011-01-01 04:53:00	13904	20110101	45300	12	...	25.0	11.0	020	29.51	30.04
5	2011-01-01 05:53:00	13904	20110101	55300	12	...	28.0	6.0	010	29.53	30.06
6	2011-01-01 06:53:00	13904	20110101	65300	12	...	29.0	7.0	360	29.57	30.10
7	2011-01-01 07:53:00	13904	20110101	75300	12	...	29.0	11.0	020	29.59	30.12
8	2011-01-01 08:53:00	13904	20110101	85300	12	...	25.0	15.0	020	29.62	30.16
9	2011-01-01 09:53:00	13904	20110101	95300	12	...	22.0	18.0	010	29.65	30.19

Source: National Oceanic & Atmospheric Administration

www.noaa.gov/climate

Reminder: `read_csv()`

- Useful keyword options
 - `names`: assigning column labels
 - `index_col`: assigning index
 - `parse_dates`: parsing datetimes
 - `na_values`: parsing NaNs

Let's practice!

PANDAS FOUNDATIONS

Statistical exploratory data analysis

PANDAS FOUNDATIONS



Dhavide Aruliah
Director of Training, Anaconda

Reminder: time series

- Index selection by date time
- Partial datetime selection
- Slicing ranges of datetimes

```
climate2010['2010-05-31 22:00:00'] # datetime
```

```
climate2010['2010-06-01'] # Entire day
```

```
climate2010['2010-04'] # Entire month
```

```
climate2010['2010-09':'2010-10'] # 2 months
```

Reminder: statistics methods

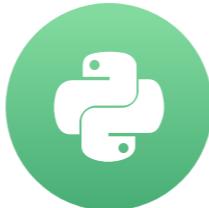
- Methods for computing statistics:
 - `describe()`: summary
 - `mean()`: average
 - `count()`: counting entries
 - `median()`: median
 - `std()`: standard deviation

Let's practice!

PANDAS FOUNDATIONS

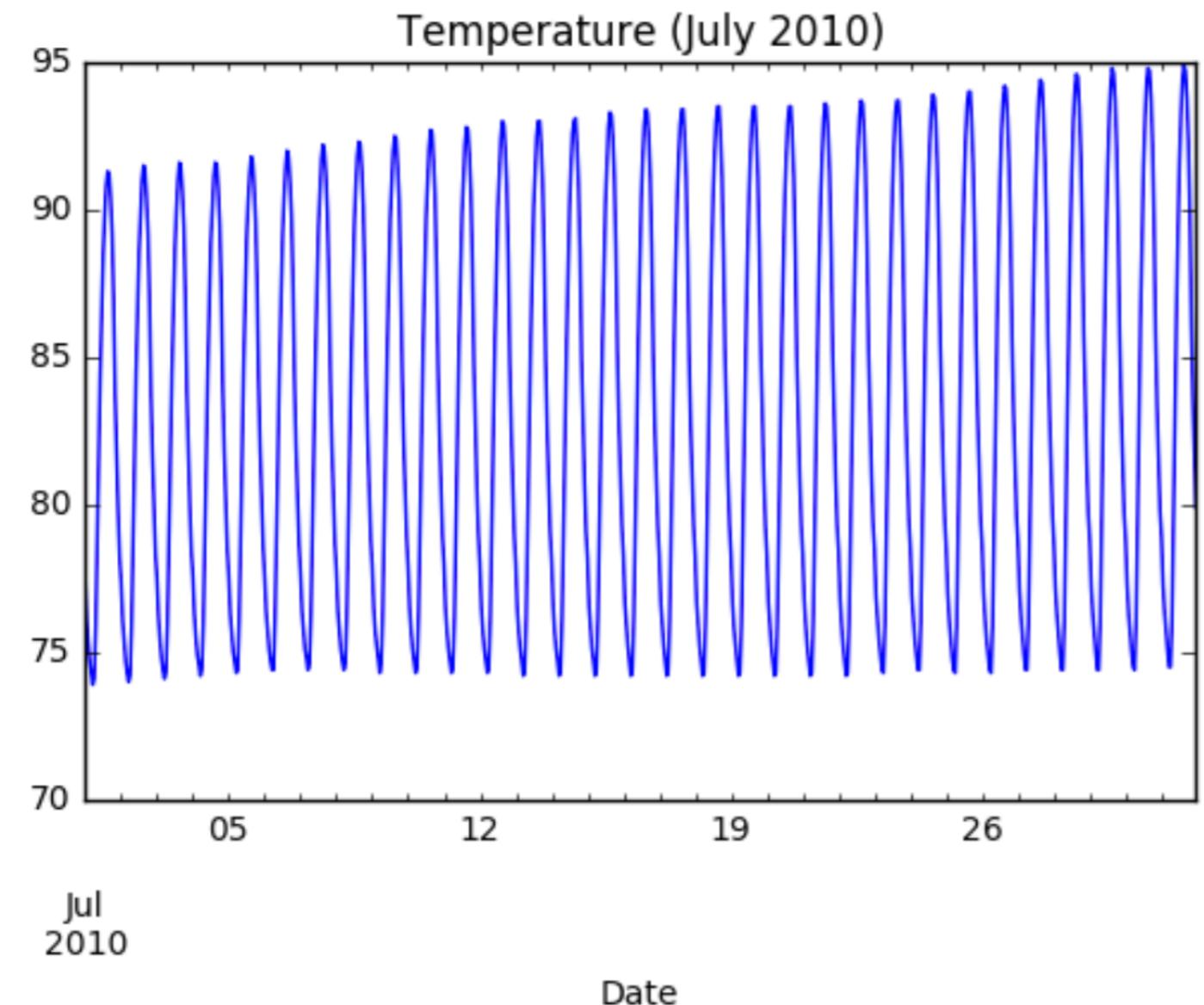
Visual exploratory data analysis

PANDAS FOUNDATIONS



Dhavide Aruliah
Director of Training, Anaconda

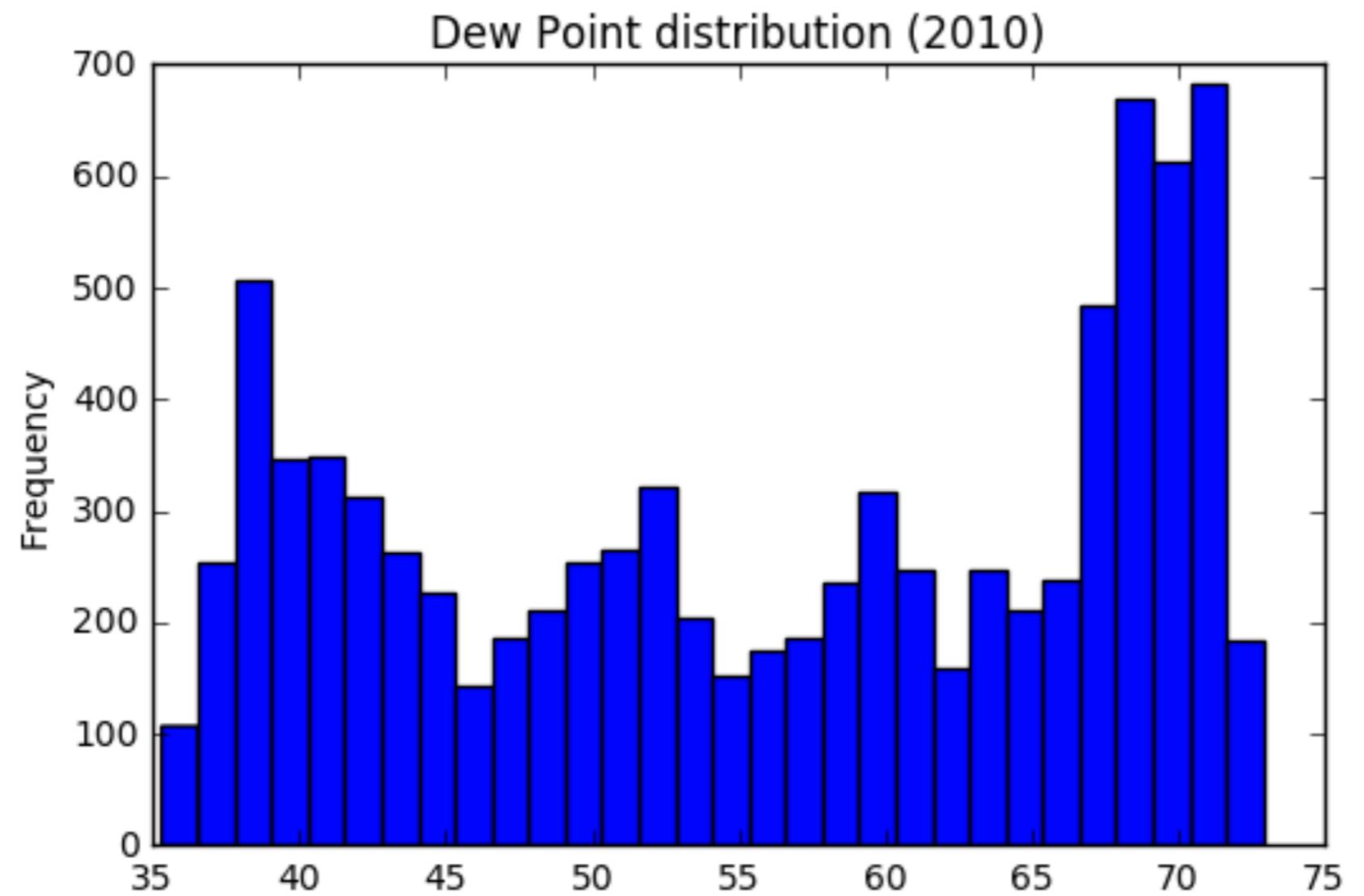
Line plots in pandas



Line plots in pandas

```
import matplotlib.pyplot as plt  
climate2010.Temperature['2010-07'].plot()  
plt.title('Temperature (July 2010)')  
plt.show()
```

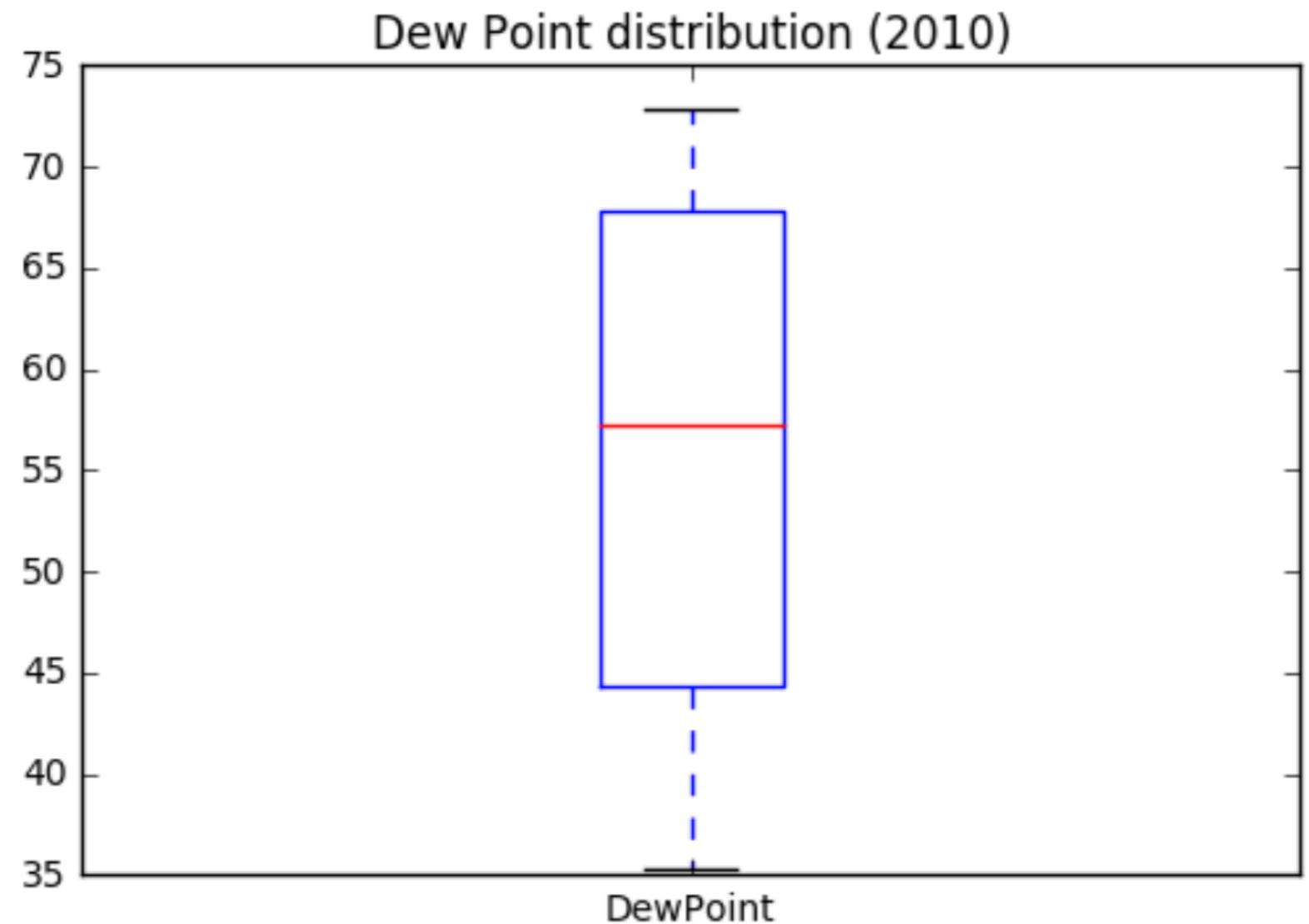
Histograms in pandas



Histograms in pandas

```
climate2010[ 'DewPoint' ].plot(kind= 'hist', bins=30)
plt.title('Dew Point distribution (2010)')
plt.show()
```

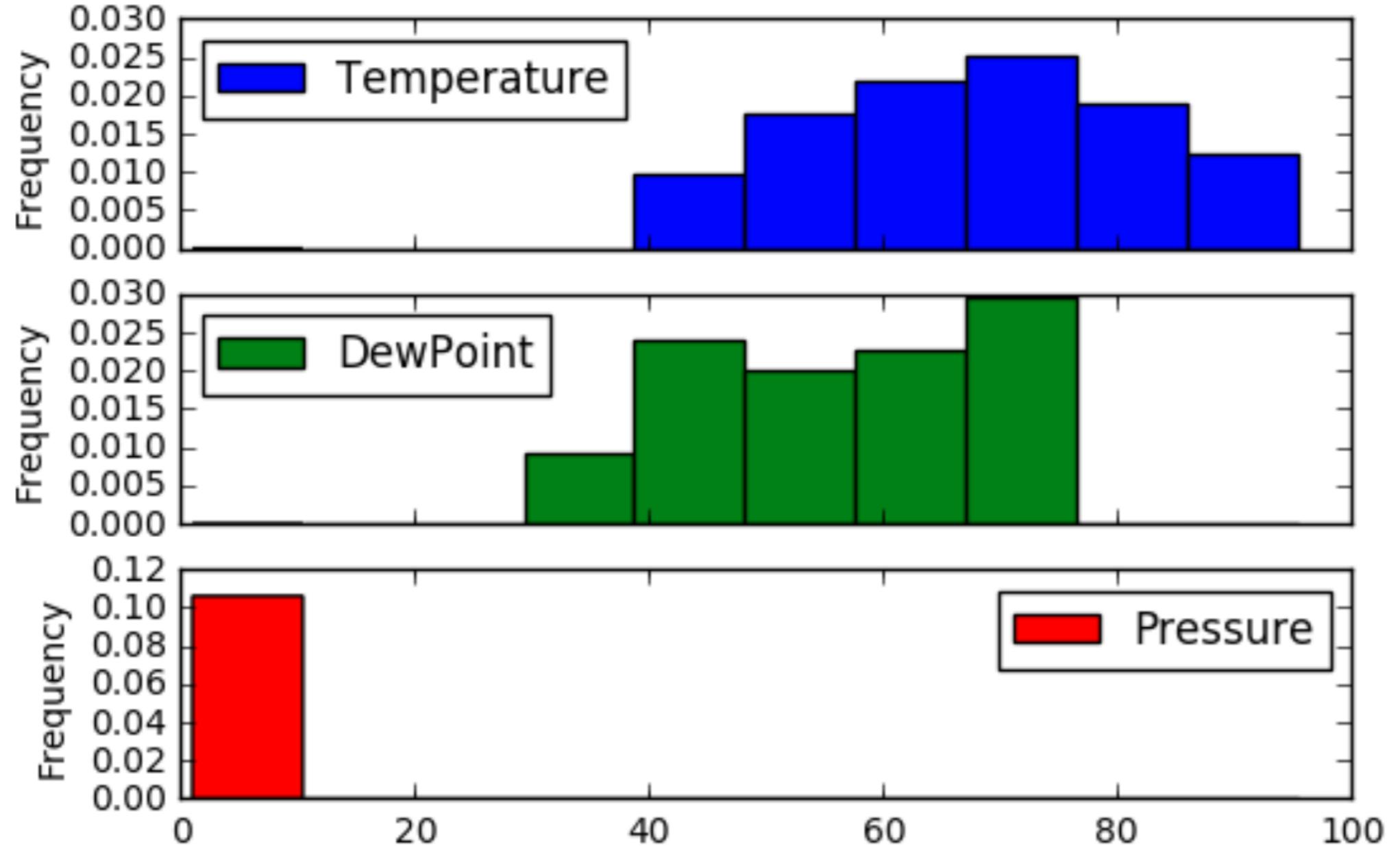
Box plots in pandas



Box plots in pandas

```
climate2010[ 'DewPoint' ].plot(kind='box')
plt.title('Dew Point distribution (2010)')
plt.show()
```

Subplots in pandas



Subplots in pandas

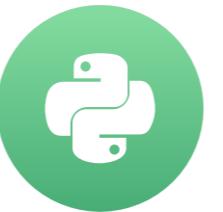
```
climate2010.plot(kind='hist', normed=True, subplots=True)  
plt.show()
```

Let's practice!

PANDAS FOUNDATIONS

Final thoughts

PANDAS FOUNDATIONS



Dhavide Aruliah

Director of Training, Anaconda

You now can...

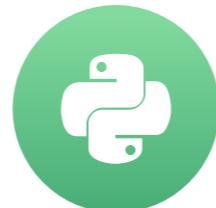
- Import many types of datasets and deal with import issues
- Export data to facilitate collaborative data science
- Perform statistical and visual EDA natively in pandas

Let's practice!

PANDAS FOUNDATIONS

Review of pandas DataFrames

PANDAS FOUNDATIONS



Dhavide Aruliah
Director of Training, Anaconda

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

```
import pandas as pd  
type(AAPL)
```

```
pandas.core.frame.DataFrame
```

```
AAPL.shape
```

```
(8514, 6)
```

```
AAPL.columns
```

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

```
type(AAPL.columns)
```

```
pandas.indexes.base.Index
```

Indexes and columns

```
AAPL.index
```

```
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)
```

```
type(AAPL.index)
```

```
pandas.tseries.index.DatetimeIndex
```

Slicing

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

	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

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

	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()

```
AAPL.head(5)
```

	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

```
AAPL.head(2)
```

	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()

```
AAPL.tail()
```

	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

```
AAPL.tail(3)
```

	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()

```
AAPL.info()
```

```
<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

- Assigning scalar value to column slice broadcasts value to each row.

```
import numpy as np  
AAPL.iloc[::3, -1] = np.nan
```

```
AAPL.head(6)
```

	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

```
AAPL.info()
```

```
<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

```
low = AAPL['Low']  
type(low)
```

```
pandas.core.series.Series
```

```
low.head()
```

```
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
```

```
lows = low.values  
type(lows)
```

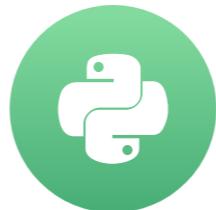
```
numpy.ndarray
```

Let's practice!

PANDAS FOUNDATIONS

Building DataFrames from scratch

PANDAS FOUNDATIONS



Dhavide Aruliah
Director of Training, Anaconda

DataFrames from CSV files

```
import pandas as pd  
users = pd.read_csv('datasets/users.csv', index_col=0)  
print(users)
```

	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)

```
import pandas as pd  
  
data = {'weekday': ['Sun', 'Sun', 'Mon', 'Mon'],  
        'city': ['Austin', 'Dallas', 'Austin', 'Dallas'],  
        'visitors': [139, 237, 326, 456],  
        'signups': [7, 12, 3, 5]}  
  
users = pd.DataFrame(data)  
print(users)
```

```
  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)

```
import pandas as pd

cities = ['Austin', 'Dallas', 'Austin', 'Dallas']

signups = [7, 12, 3, 5]

visitors = [139, 237, 326, 456]

weekdays = ['Sun', 'Sun', 'Mon', 'Mon']

list_labels = ['city', 'signups', 'visitors', 'weekday']

list_cols = [cities, signups, visitors, weekdays]

zipped = list(zip(list_labels, list_cols))
```

DataFrames from dict (3)

```
print(zipped)
```

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

```
data = dict(zipped)
users = pd.DataFrame(data)
print(users)
```

```
   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

```
users['fees'] = 0 # Broadcasts to entire column  
print(users)
```

	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

```
import pandas as pd  
  
heights = [ 59.0, 65.2, 62.9, 65.4, 63.7, 65.7, 64.1 ]  
data = {'height': heights, 'sex': 'M'}  
results = pd.DataFrame(data)  
print(results)
```

```
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

```
results.columns = ['height (in)', 'sex']
results.index = ['A', 'B', 'C', 'D', 'E', 'F', 'G']
print(results)
```

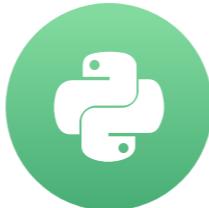
	height (in)	sex
A	59.0	M
B	65.2	M
C	62.9	M
D	65.4	M
E	63.7	M
F	65.7	M
G	64.1	M

Let's practice!

PANDAS FOUNDATIONS

Importing & exporting data

PANDAS FOUNDATIONS



Dhavide Aruliah
Director of Training, Anaconda

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  
...
```

¹ Source: SILSO, Daily total sunspot number
(<http://www.sidc.be/silso/info&ssntodaily>)

Datasets from CSV files

```
import pandas as pd  
filepath = 'ISSN_D_tot.csv'  
sunspots = pd.read_csv(filepath)  
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

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

```
   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

```
sunspots = pd.read_csv(filepath, header=None)  
sunspots.iloc[10:20, :]
```

	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

```
col_names = ['year', 'month', 'day', 'dec_date',
             'sunspots', 'definite']
sunspots = pd.read_csv(filepath, header=None,
                       names=col_names)
sunspots.iloc[10:20, :]
```

	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)

```
sunspots = pd.read_csv(filepath, header=None,  
                      names=col_names, na_values='-1')  
sunspots.iloc[10:20, :]
```

	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)

```
sunspots = pd.read_csv(filepath, header=None,  
                      names=col_names, na_values=' -1 ')  
sunspots.iloc[10:20, :]
```

	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)

```
sunspots = pd.read_csv(filepath, header=None,  
                      names=col_names, na_values={'sunspots':[-1]})  
sunspots.iloc[10:20, :]
```

	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

```
sunspots = pd.read_csv(filepath, header=None,  
                      names=col_names, na_values={'sunspots':[-1]},  
                      parse_dates=[[0, 1, 2]])  
sunspots.iloc[10:20, :]
```

	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

```
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

```
sunspots.index = sunspots['year_month_day']
sunspots.index.name = 'date'
sunspots.info()
```

```
<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

```
cols = ['sunspots', 'definite']
sunspots = sunspots[cols]
sunspots.iloc[10:20, :]
```

```
      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

```
out_csv = 'sunspots.csv'  
sunspots.to_csv(out_csv)
```

```
out_tsv = 'sunspots.tsv'  
sunspots.to_csv(out_tsv, sep='\t')
```

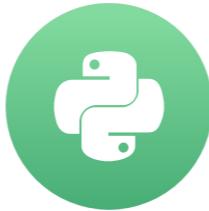
```
out_xlsx = 'sunspots.xlsx'  
sunspots.to_excel(out_xlsx)
```

Let's practice!

PANDAS FOUNDATIONS

Plotting with pandas

PANDAS FOUNDATIONS



Dhavide Aruliah

Director of Training, Anaconda

AAPL stock data

```
import pandas as pd
import matplotlib.pyplot as plt
aapl = pd.read_csv('aapl.csv', index_col='date',
                    parse_dates=True)
aapl.head(6)
```

	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)

```
close_arr = aapl['close'].values  
type(close_arr)
```

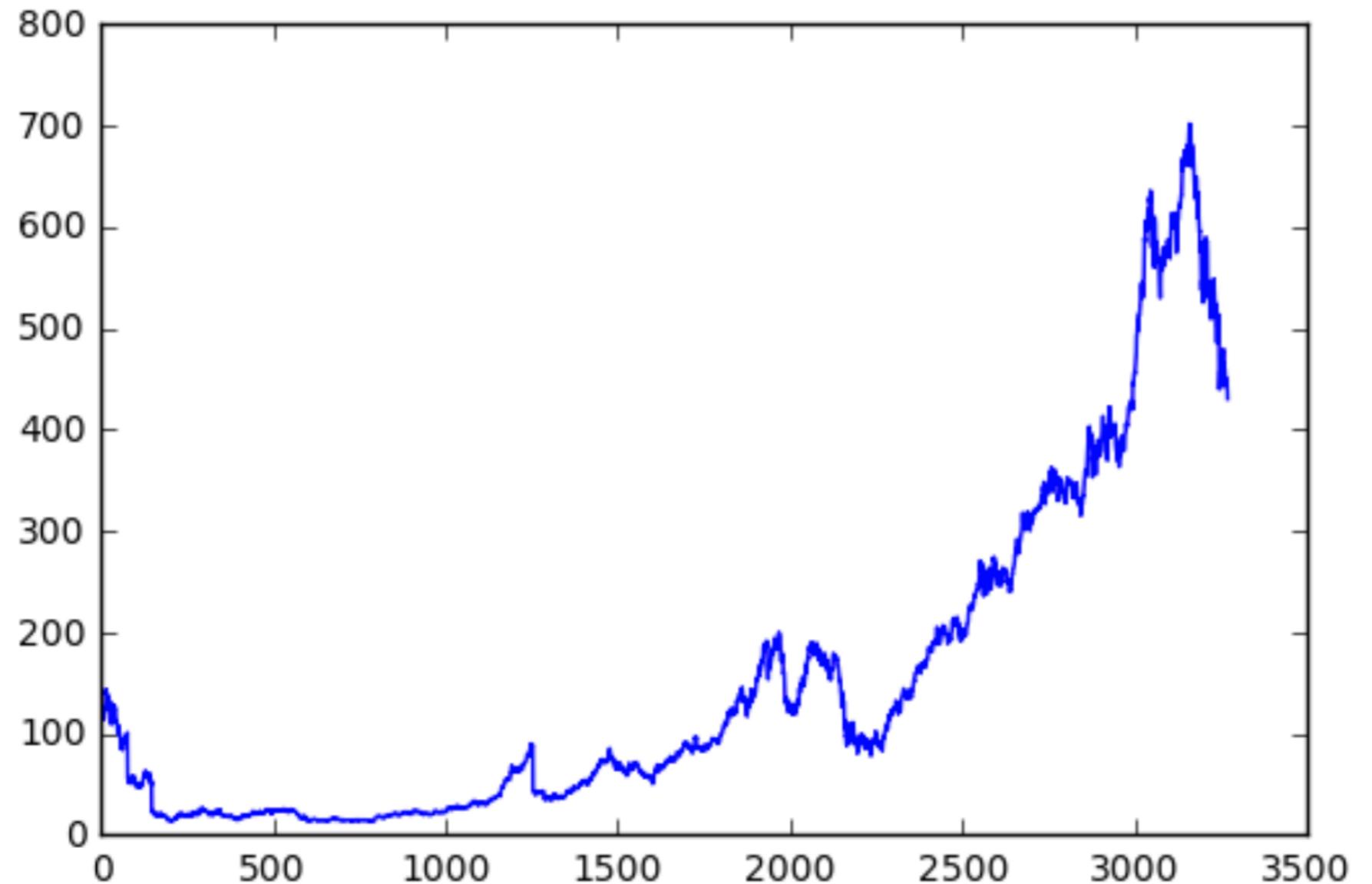
```
numpy.ndarray
```

```
plt.plot(close_arr)
```

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

```
plt.show()
```

Plotting arrays (matplotlib)



Plotting Series (matplotlib)

```
close_series = aapl['close']
type(close_series)
```

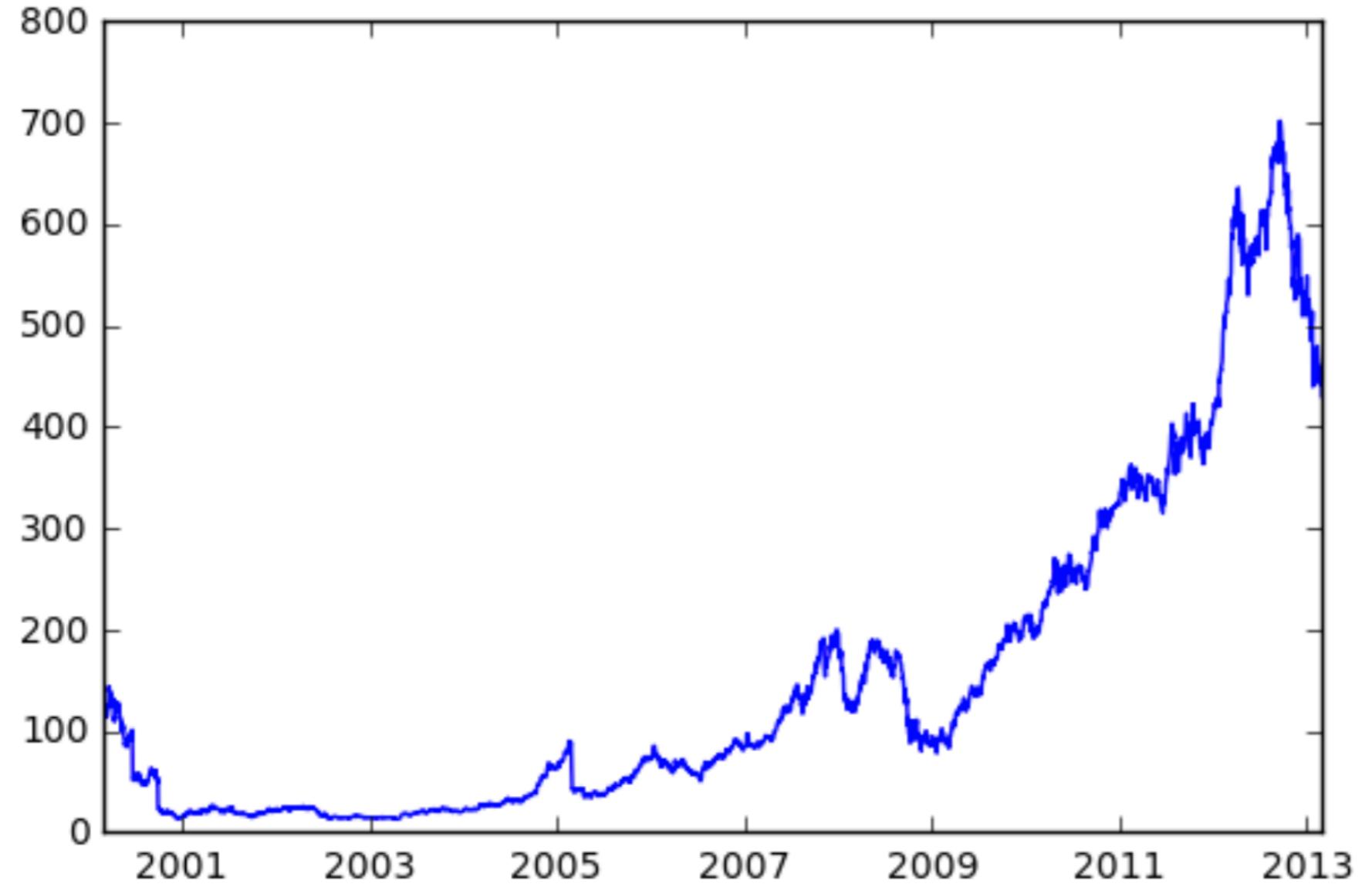
```
pandas.core.series.Series
```

```
plt.plot(close_series)
```

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

```
plt.show()
```

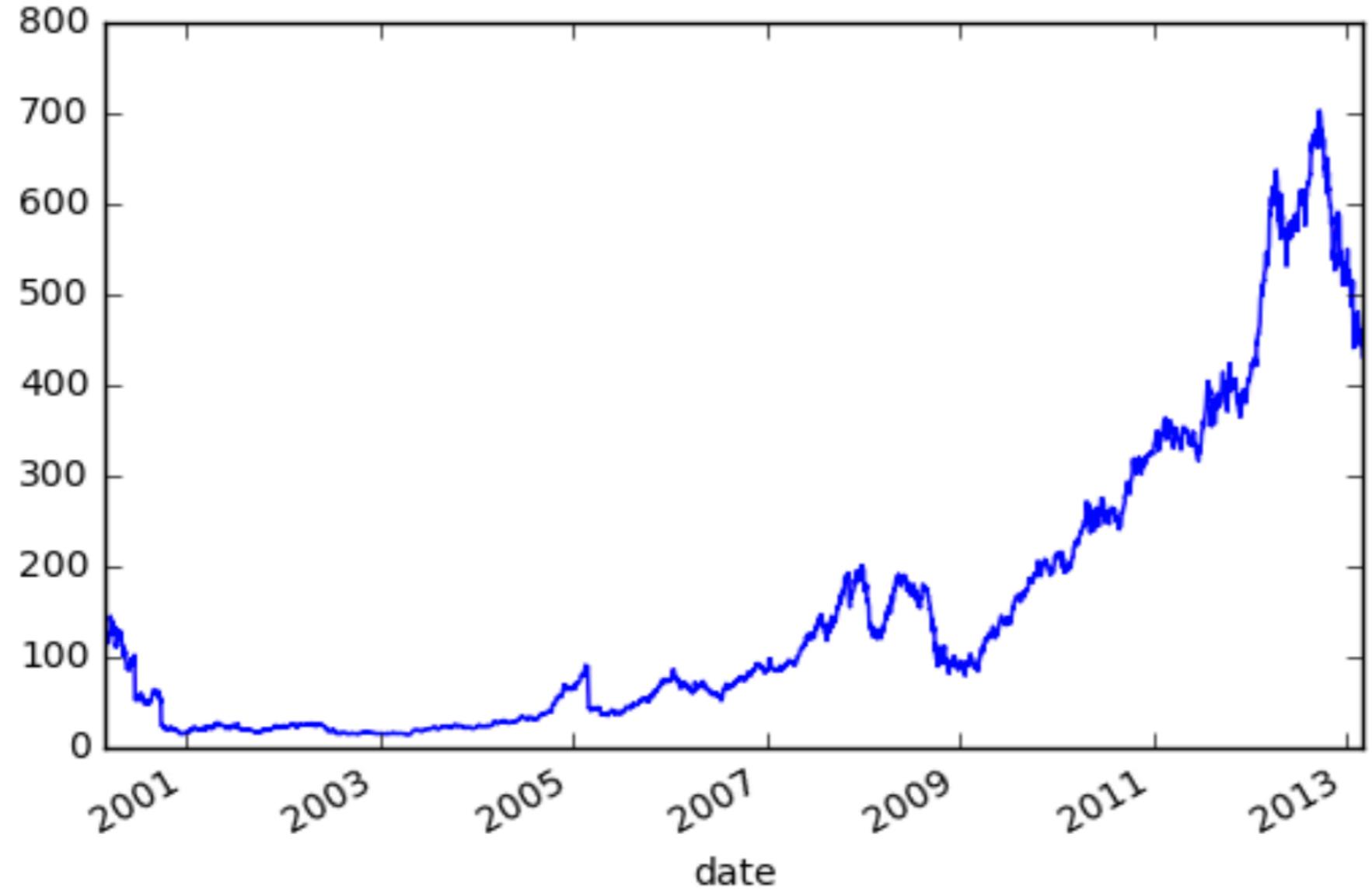
Plotting Series (matplotlib)



Plotting Series (pandas)

```
close_series.plot() # plots Series directly  
plt.show()
```

Plotting Series (pandas)



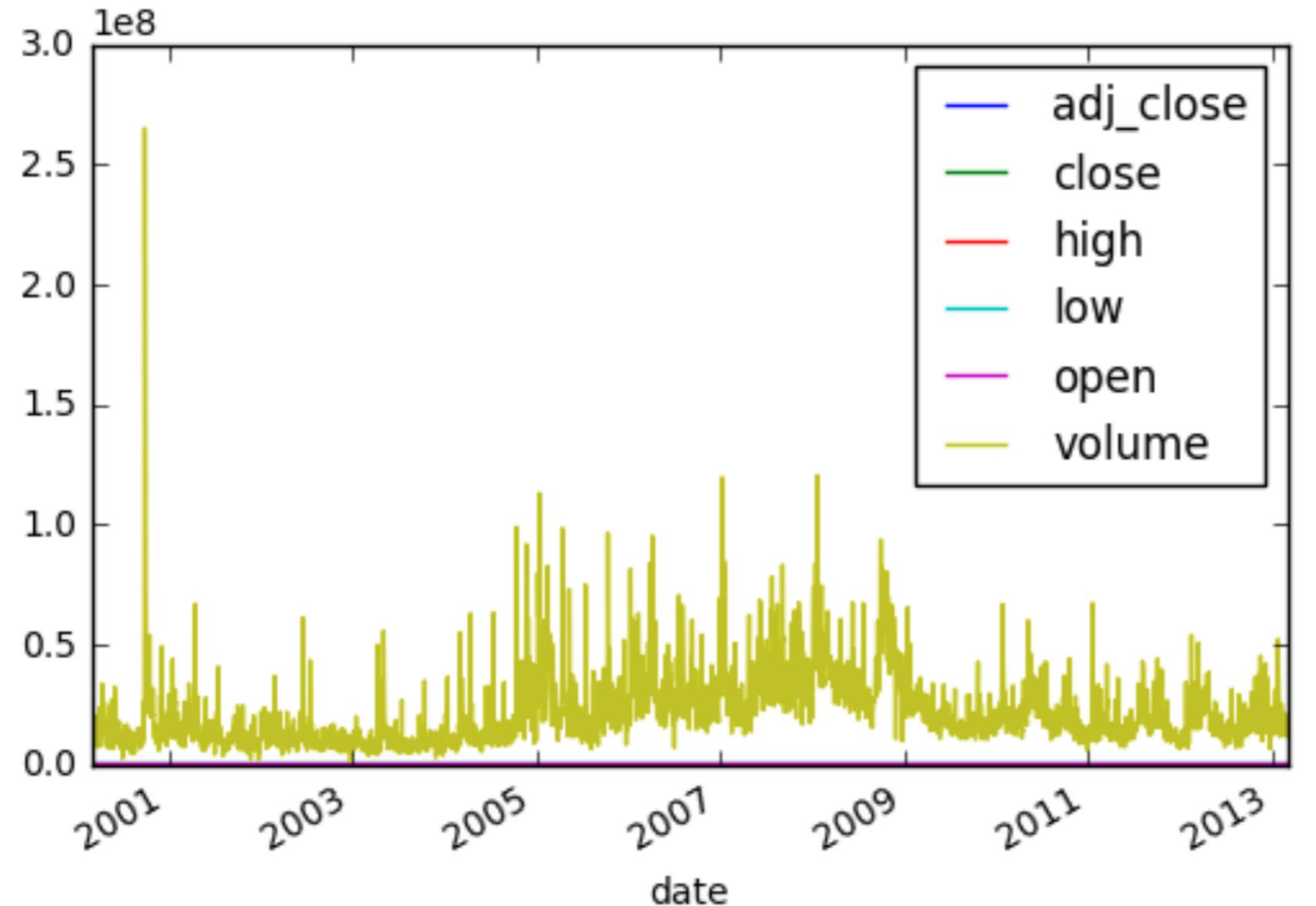
Plotting DataFrames (pandas)

```
aapl.plot() # plots all Series at once
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x118039b38>
```

```
plt.show()
```

Plotting DataFrames (pandas)



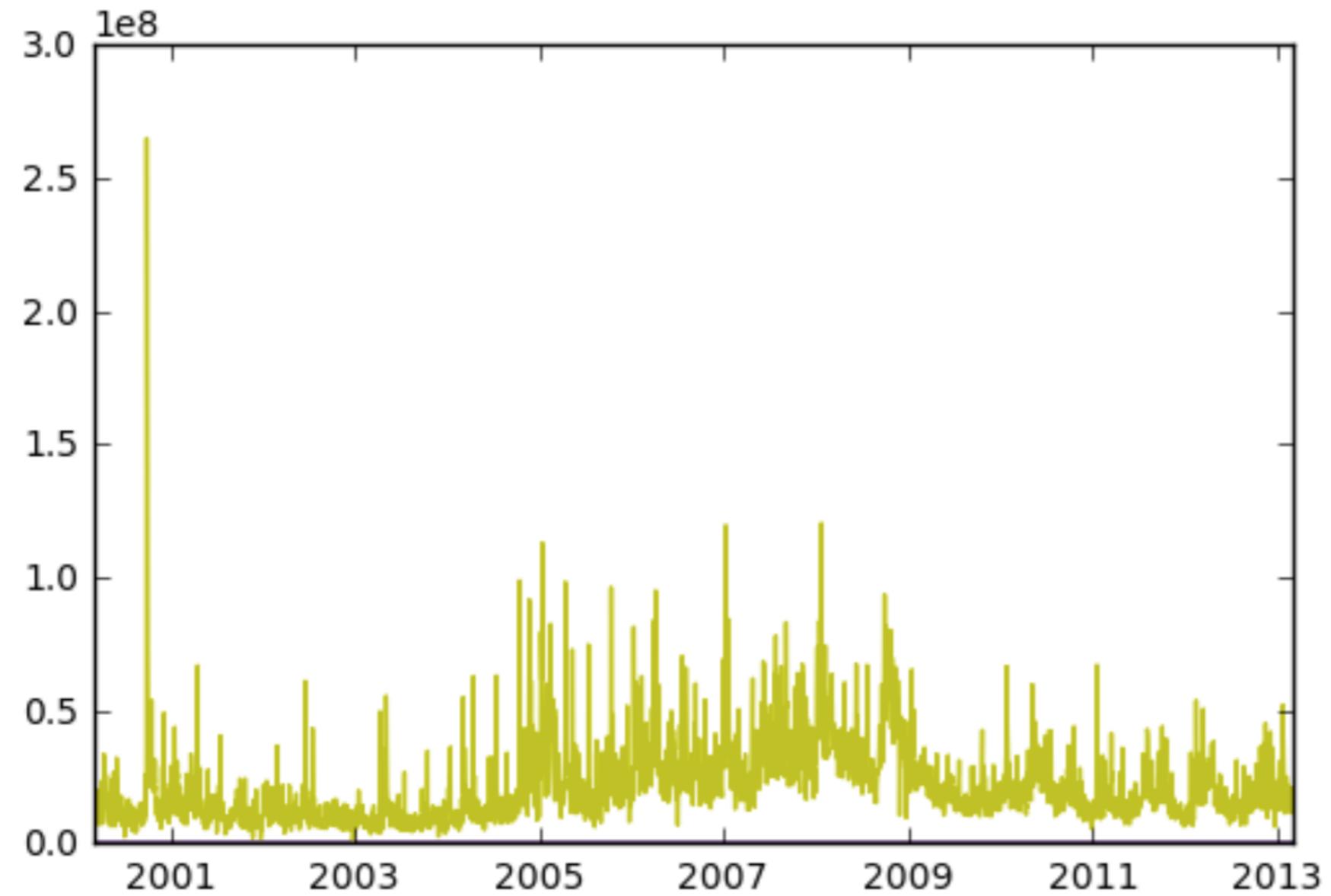
Plotting DataFrames (matplotlib)

```
plt.plot(aapl) # plots all columns at once
```

```
<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>
```

```
plt.show()
```

Plotting DataFrames (matplotlib)



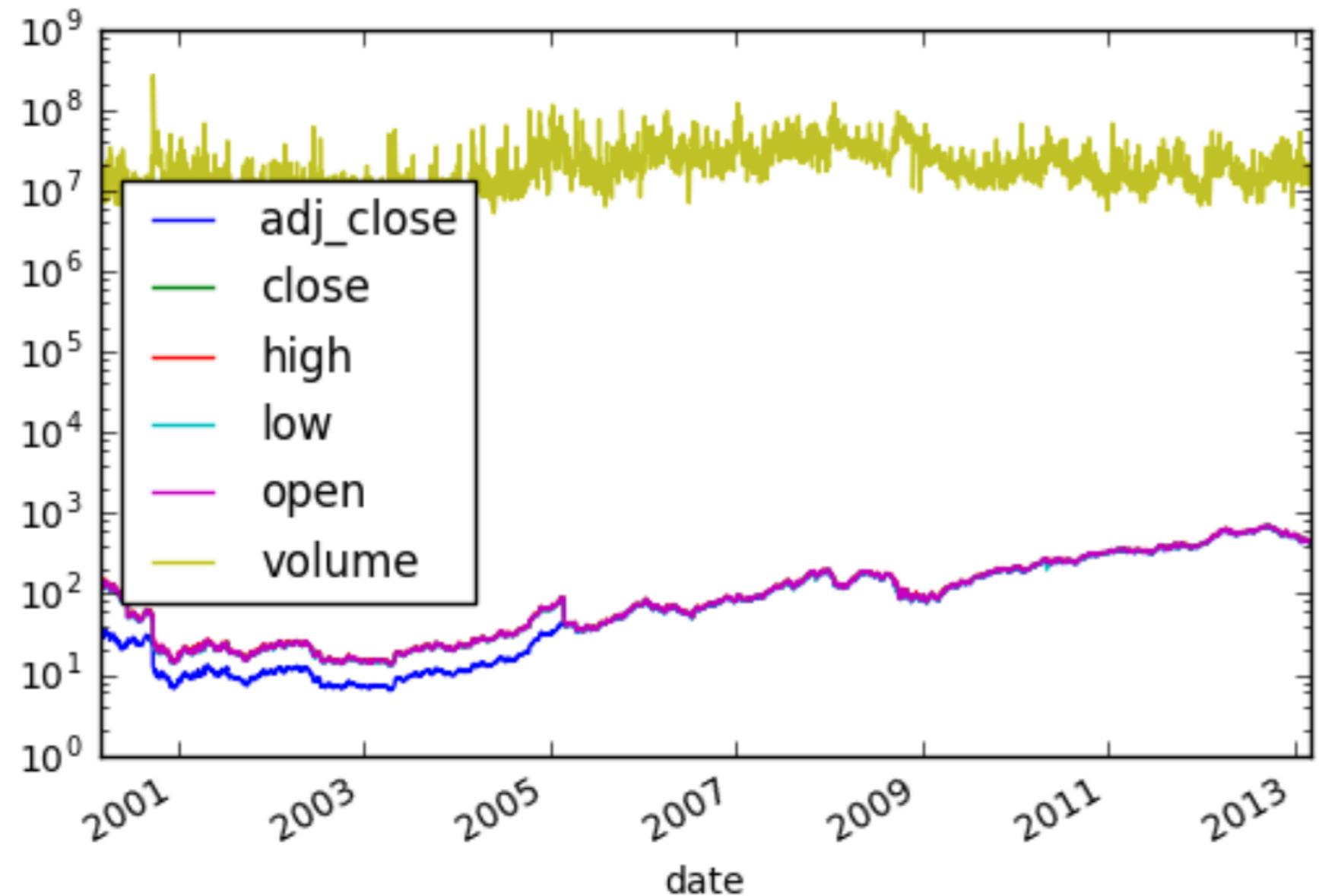
Fixing scales

```
aapl.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x118afe048>
```

```
plt.yscale('log') # logarithmic scale on vertical axis  
plt.show()
```

Fixing scales



Customizing plots

```
aapl['open'].plot(color='b', style='.-', legend=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11a17db38>
```

```
aapl['close'].plot(color='r', style='.', legend=True)
```

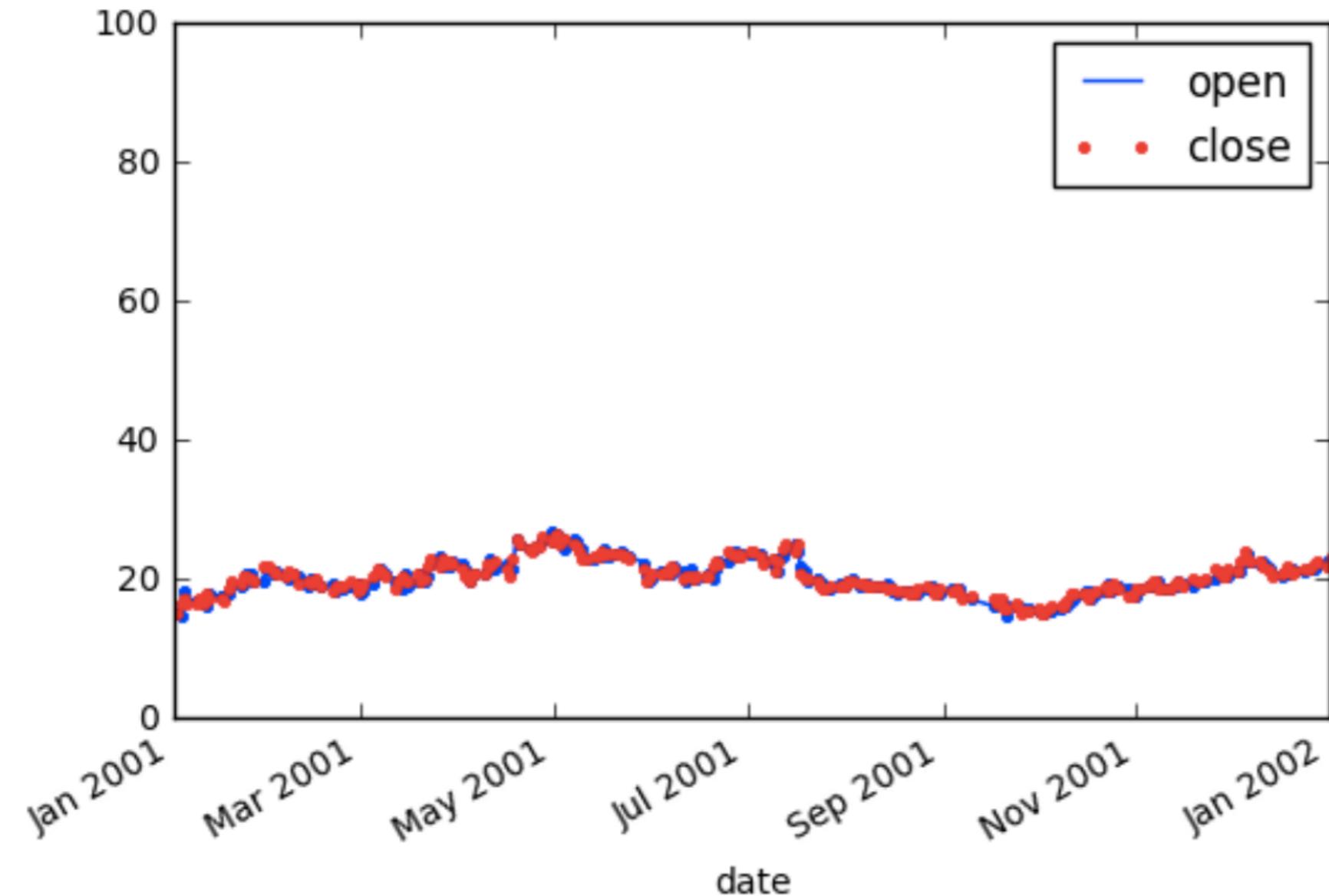
```
<matplotlib.axes._subplots.AxesSubplot at 0x11a17db38>
```

```
plt.axis((2001, 2002, 0, 100))
```

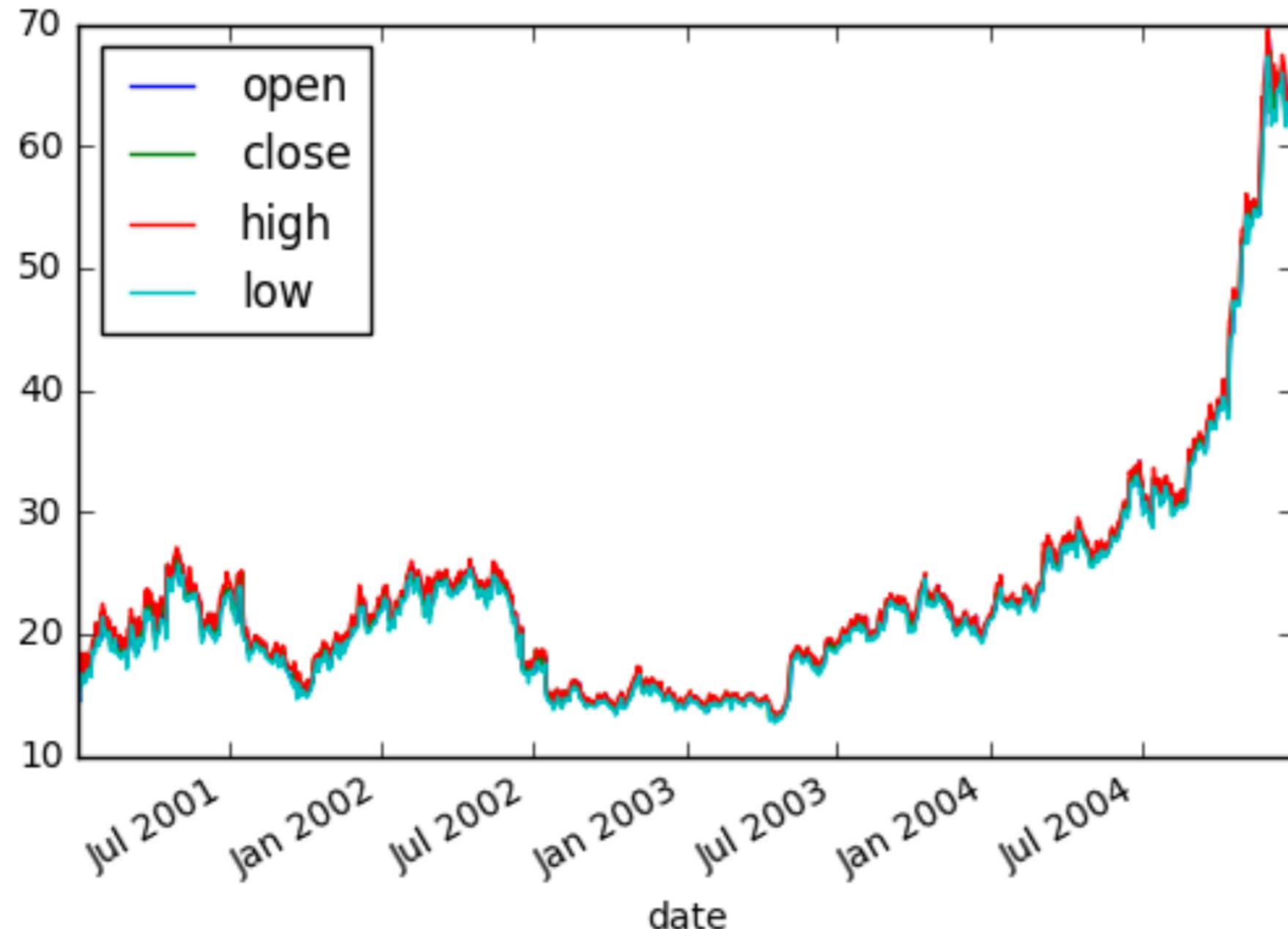
```
'2001', '2002', 0, 100)
```

```
plt.show()
```

Customizing plots



Saving plots



Saving plots

```
aapl.loc['2001':'2004',['open', 'close', 'high',  
'low']].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11ab42978>
```

```
plt.savefig('aapl.png')  
plt.savefig('aapl.jpg')  
plt.savefig('aapl.pdf')  
plt.show()
```

Let's practice!

PANDAS FOUNDATIONS