HUGE

Hello

Handling data with python

August 11, 2018

1. Libraries
2. Numpy
3. Pandas

Agenda.

H

Libraries

- Numpy
- Pandas
- Matplotlib

Numpy

Allows us to handle arrays. Essentially an array is a table of elements, all of the same type.

(1, 2, 1)

An array has axes. An axis can be thinking as a dimension of an array. The following array has two axes:

([1., 0., 0.],
[0., 1., 2.]])

The first axis has a length of 2, and the second one a length of three.

Actually, this array can be thought as an array of arrays which has only one dimension of length 2 and each element is another array.

Numpy arrays

Numpy arrays are objects of the class ndarray, which has the following properties:

- **ndarray.ndim**: Number of axes
- **ndarray.shape**: An array with the length on each dimension.
- **ndarray.size**: Total number of elements.
- **ndarray.dtype**: Type of the elements
- **ndarray.itemsize**: Size in bytes of each element.
- **ndarray.data**: Actual elements.

Numpy arrays

```
>>> import numpy as np

>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
```

Numpy arrays

```
>>> a.itemsize
8
>>> a.size
15
>>> type(a)
<type 'numpy.ndarray'>
>>> b = np.array([6, 7, 8])
>>> b
array([6, 7, 8])
>>> type(b)
<type 'numpy.ndarray'>
```

Numpy arrays

```
>>> b = np.array([(1.5, 2, 3), (4, 5, 6)])
array([[ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])

>>> c = np.array( [ [1, 2], [3, 4] ], dtype=complex )
array([[ 1.+0.j,  2.+0.j],
       [ 3.+0.j,  4.+0.j]])
```

Numpy arrays

```
# Array from tuples
>>> b = np.array([(1.5, 2, 3), (4, 5, 6)])
array([[ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])

# Specifying the type of the array
>>> c = np.array( [ [1, 2], [3, 4] ], dtype=complex )
array([[ 1.+0.j,  2.+0.j],
       [ 3.+0.j,  4.+0.j]])
```

Numpy arrays

```
# Array from tuples
>>> b = np.array([(1.5, 2, 3), (4, 5, 6)])
array([[ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])

# Specifying the type of the array
>>> c = np.array( [ [1, 2], [3, 4] ], dtype=complex )
array([[ 1.+0.j,  2.+0.j],
       [ 3.+0.j,  4.+0.j]])
```

Numpy arrays

```
>>> np.zeros( (3,4) )
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])

>>> np.ones( (2,3,4), dtype=np.int16 ) # np.ones(shape=(2,3,4)) produces the same array
array([[[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
        [ 1, 1, 1, 1]],
       [[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
        [ 1, 1, 1, 1]]], dtype=int16)
```

Numpy arrays

```
np.empty( (2,3) ) # Random values, depends on the state of the memory
array([[  3.73603959e-262,   6.02658058e-154,   6.55490914e-260],
       [  5.30498948e-313,   3.14673309e-307,   1.00000000e+000]])

# To create sequences of numbers from 10 to 30 - 1 in steps of 5 units
>>> np.arange( 10, 30, 5 )
array([10, 15, 20, 25])

# To create sequences of numbers from 0 to 2, equally divided in 9 steps
>>> np.linspace( 0, 2, 9 )
array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ,  1.25,  1.5 ,  1.75,  2.  ])
```

Numpy arrays

```
>>> a = np.arange(6)         # Same that np.arange(0,6) or np.arrange(0,6,1)
>>> print(a)
[0 1 2 3 4 5]

>>> b = np.arange(12).reshape(4,3)  # Converting [0, … , 11] to a two dimensional
>>> print(b)                         # array of shape 4 x 3 ( 4 x 3 = 12 same # of els)
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

Array operations

```
>>> a = np.arange(6)          # Same that np.arange(0,6) or np.arrange(0,6,1)
>>> print(a)
[0 1 2 3 4 5]

>>> b = np.arange(12).reshape(4,3)  # Converting [0, … , 11] to a two dimensional
>>> print(b)                         # array of shape 4 x 3 ( 4 x 3 = 12 same # of els)
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

Array operations

```
>>> a = np.array( [20,30,40,50] )
>>> b = np.arange( 4 )

>>> c = a - b # Subtraction

>>> c = a + b # Sum

>>> c = a ** b # Exponential

>>> c = np.sin(b) # Trigonometric expressions
```

Array operations

```
>>> c = a @ b # Matrix operations

>>> c = np.dot(a, b) # Same as before, multiplication between arrays

>>> a = np.ones((2,3), dtype=int)
>>> a *= 3
array([[3, 3, 3],
       [3, 3, 3]])
```

Metrics of an array

```
>>> a = np.random.random((2,3))
>>> a
array([[ 0.18626021,  0.34556073,  0.39676747],
       [ 0.53881673,  0.41919451,  0.6852195 ]])
>>> a.sum()
2.5718191614547998
>>> a.min()
0.1862602113776709
>>> a.max()
0.6852195003967595
```

By default, these operations apply to the array as though it were a list of numbers, regardless of its shape.

Metrics of an array

```
>>> b = np.arange(12).reshape(3,4)
>>> b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])

>>> b.sum(axis=0)                            # sum of each column
array([12, 15, 18, 21])

>>> b.min(axis=1)                            # min of each row
array([0, 4, 8])
```

Read a csv

Download:
https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv

```
my_data = np.genfromtxt('winequality-red.csv', delimiter=',')
# Make sure that the delimiter is a comma

my_data = np.genfromtxt('winequality-red.csv', delimiter=';')
# Text will be read as nan
```

Indexing

```
>>> my_data[2]
array([ 7.8   ,  0.88  ,  0.    ,  2.6   ,  0.098 , 25.    , 67.    ,
        0.9968,  3.2   ,  0.68  ,  9.8   ,  5.    ])

>>> my_data[2:5]
array([[7.800e+00, 8.800e-01, …, 9.800e+00, 5.000e+00],
       [7.800e+00, 7.600e-01, …, 9.800e+00, 5.000e+00],
       [1.120e+01, 2.800e-01, …, 9.800e+00, 6.000e+00]])

>>> my_data[2]
array([ 5.    ,  9.8   ,  0.68  ,  3.2   ,  0.9968, 67.    , 25.    ,
        0.098 ,  2.6   ,  0.    ,  0.88  ,  7.8   ])
```

Indexing

```
>>> my_data[2, 1]
0.88

>>> my_data[0:5, 1]
array([0.88, 0.76, 0.28])

>>> my_data[-1]                          # the last row. Equivalent to my_data[-1,:]

>>> my_data.shape
(1600, 12)
```

Indexing

```
>>> my_data[1,...]                        # same as my_data[1,:,:] or my_data[1]
array([ 7.4   ,  0.7   ,  0.    ,  1.9   ,  0.076 , 11.    , 34.    ,
        0.9978,  3.51  ,  0.56  ,  9.4   ,  5.    ])

>>> my_data[...,2]                        # same as my_data[:,:,2]
array([ nan, 0.  , 0.  , ..., 0.13, 0.12, 0.47])
```

More operations

```
>>> np.floor(10*np.random.random((3,4)))
array([[ 2.,  8.,  0.,  6.],
       [ 4.,  5.,  1.,  1.],
       [ 8.,  9.,  3.,  6.]])
>>> a.ravel()  # returns the array, flattened
array([ 2.,  8.,  0.,  6.,  4.,  5.,  1.,  1.,  8.,  9.,  3.,  6.])
>>> a.T  # returns the array, transposed
array([[ 2.,  4.,  8.],
       [ 8.,  5.,  9.],
       [ 0.,  1.,  3.],
       [ 6.,  1.,  6.]])
```

More operations

```
>>> np.floor(10*np.random.random((3,4)))
array([[ 2.,  8.,  0.,  6.],
       [ 4.,  5.,  1.,  1.],
       [ 8.,  9.,  3.,  6.]])
>>> a.ravel()  # returns the array, flattened
array([ 2.,  8.,  0.,  6.,  4.,  5.,  1.,  1.,  8.,  9.,  3.,  6.])
>>> a.T  # returns the array, transposed
array([[ 2.,  4.,  8.],
       [ 8.,  5.,  9.],
       [ 0.,  1.,  3.],
       [ 6.,  1.,  6.]])
```

More operations

```
>>> a
array([[ 2.,  8.,  0.,  6.],
       [ 4.,  5.,  1.,  1.],
       [ 8.,  9.,  3.,  6.]])

>>> a.resize((2,6))
array([[ 2.,  8.,  0.,  6.,  4.,  5.],
       [ 1.,  1.,  8.,  9.,  3.,  6.]])
```

More operations

```
>>> a
array([[ 2.,  8.,  0.,  6.],
       [ 4.,  5.,  1.,  1.],
       [ 8.,  9.,  3.,  6.]])

>>> a.resize((2,6))
array([[ 2.,  8.,  0.,  6.,  4.,  5.],
       [ 1.,  1.,  8.,  9.,  3.,  6.]])
```

Combining arrays

```
>>> b = np.floor(10*np.random.random((2,2)))
>>> b
array([[ 1.,  8.],
       [ 0.,  4.]])
>>> np.vstack((a,b))
array([[ 8.,  8.],
       [ 0.,  0.],
       [ 1.,  8.],
       [ 0.,  4.]])
>>> np.hstack((a,b))
array([[ 8.,  8.,  1.,  8.],
       [ 0.,  0.,  0.,  4.]])
```

Copies and Views

```
>>> a = np.arange(12)
>>> b = a                # no new object is created
>>> b is a               # a and b are two names for the same ndarray object
True

>>> b.shape = 3,4       # changes the shape of a
>>> a.shape
(3, 4)
```

H

Copies and Views

```
# Different array objects can share the same data. The view method creates a new array
object that looks at the same data.
>>> c = a.view()
>>> c is a
False

>>> c[0,4] = 1234                    # a's data changes
>>> a
array([[   0,    1,    2,    3],
       [1234,    5,    6,    7],
       [   8,    9,   10,   11]])
```

Copies and Views

```
>>> s = a[ : , 1:3] # spaces added for clarity; could also be written "s = a[:,1:3]"
>>> s[:] = 10       # s[:] is a view of s.
>>> a
array([[   0,   10,   10,    3],
       [1234,   10,   10,    7],
       [   8,   10,   10,   11]])
```

Copies and Views

```
>>> d = a.copy()                       # a new array object with new data is created
>>> d is a
False
>>> d.base is a                        # d doesn't share anything with a
False
>>> d[0,0] = 9999
>>> a
array([[   0,   10,   10,    3],
       [1234,   10,   10,    7],
       [   8,   10,   10,   11]])
```

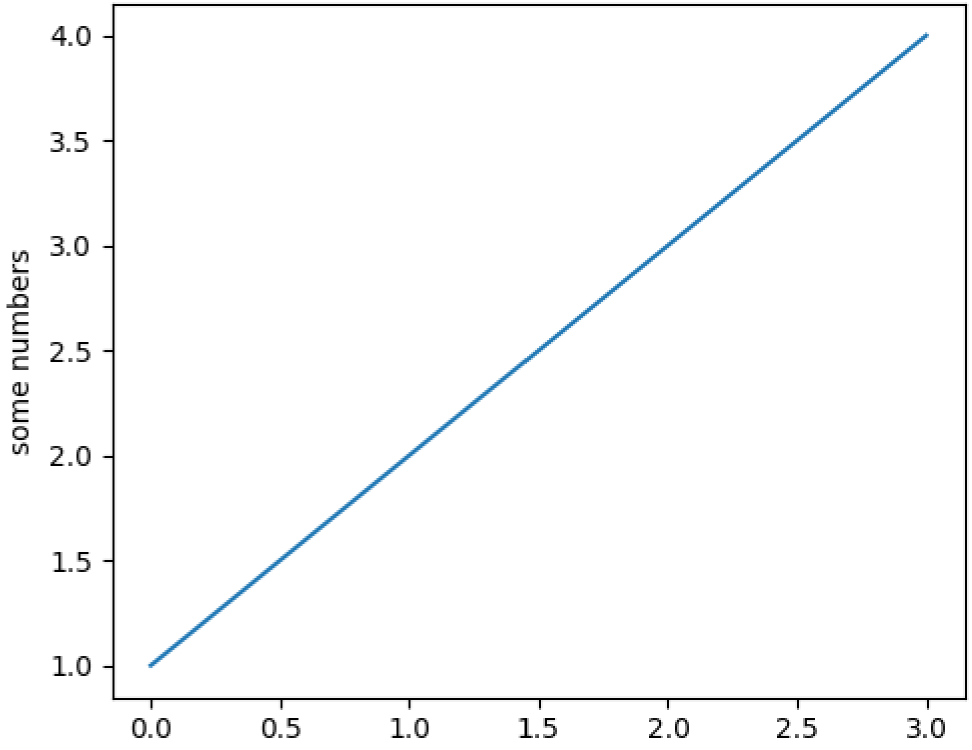
Matplotlib: Pyplot

matplotlib.pyplot is a collection of command style functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

In matplotlib.pyplot various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area…

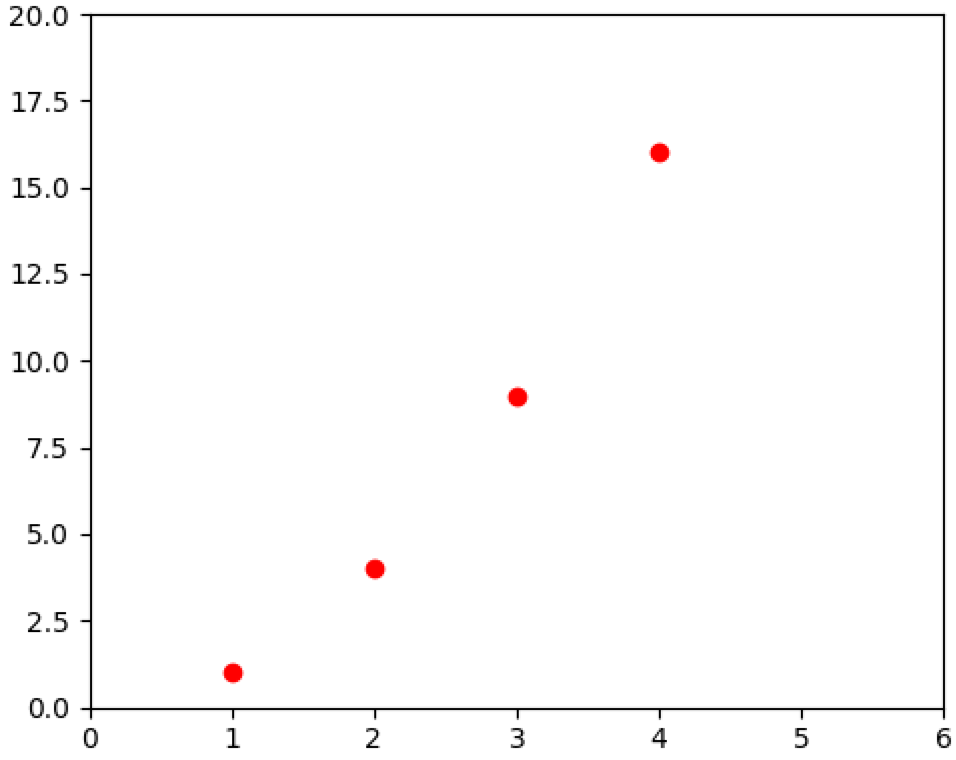
First plot

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4])
plt.ylabel('some numbers')
plt.show()
```



Plots

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4], [1,4,9,16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```



Plot properties

Property	Value Type
alpha	float
animated	[True | False]
antialiased or aa	[True | False]
clip_box	a matplotlib.transform.Bbox instance
clip_on	[True | False]
clip_path	a Path instance and a Transform instance, a Patch
color or c	any matplotlib color
contains	the hit testing function
dash_capstyle	['butt' | 'round' | 'projecting']
dash_joinstyle	['miter' | 'round' | 'bevel']
dashes	sequence of on/off ink in points
data	(np.array xdata, np.array ydata)
figure	a matplotlib.figure.Figure instance
label	any string
linestyle or ls	['-' | '--' | '-.' | ':' | 'steps' | ...]
linewidth or lw	float value in points
lod	[True | False]
marker	['+' | ',' | '.' | '1' | '2' | '3' | '4']
markeredgecolor or mec	any matplotlib color
markeredgewidth or mew	float value in points
markerfacecolor or mfc	any matplotlib color
markersize or ms	float
markevery	[None | integer | (startind, stride)]
picker	used in interactive line selection
pickradius	the line pick selection radius
solid_capstyle	['butt' | 'round' | 'projecting']
solid_joinstyle	['miter' | 'round' | 'bevel']
transform	a matplotlib.transforms.Transform instance
visible	[True | False]
xdata	np.array
ydata	np.array
zorder	any number

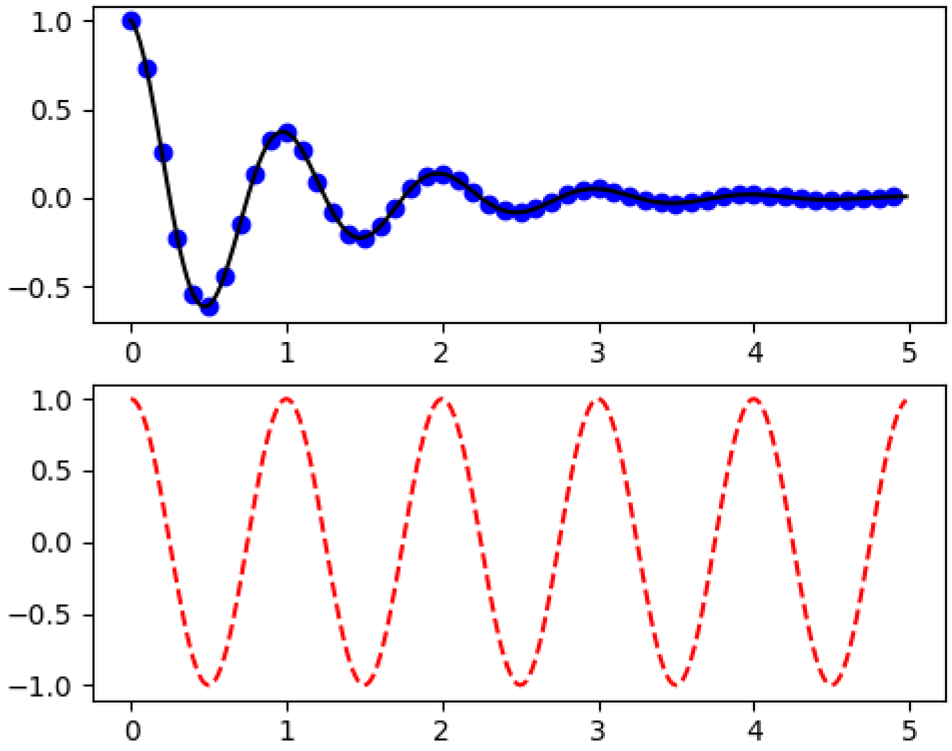
Multiple figures and axes

```
def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure(1)
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```

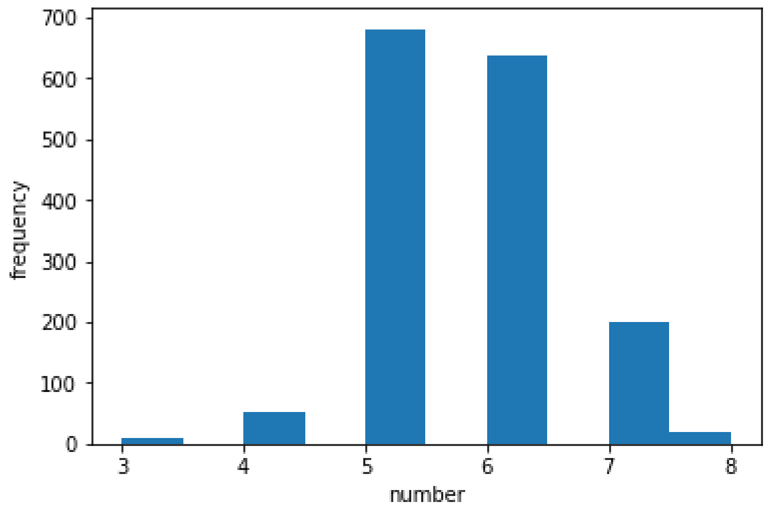


Multiple figures and axes

```
import matplotlib.pyplot as plt

# the histogram of the data
n, bins, patches = plt.hist(my_data[:, -1], bins = 9)

plt.xlabel('number')
plt.ylabel('frequency’)
```



Pandas

Allows us to handle data structures and data analysis tools for the Python programming language.

Creating a dataframe

```
import pandas as pd
# The inital set of baby
names and bith rates
names = ['Bob','Jessica','Mary','John','Mel']
births = [968, 155, 77, 578, 973]

list(zip(names, births))

df = pd.DataFrame(
 data = BabyDataSet,
 columns=['Names', 'Births’]
)
```

	Names	Births
0	Bob	968
1	Jessica	155
2	Mary	77
3	John	578
4	Mel	973

Reading data

```
df = pd.read_csv('winequality-red.csv', sep=';’)

df
```

	7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5
0	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
1	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
2	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
3	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
4	7.4	0.660	0.00	1.8	0.075	13.0	40.0	0.99780	3.51	0.56	9.4	5
5	7.9	0.600	0.06	1.6	0.069	15.0	59.0	0.99640	3.30	0.46	9.4	5
6	7.3	0.650	0.00	1.2	0.065	15.0	21.0	0.99460	3.39	0.47	10.0	7
7	7.8	0.580	0.02	2.0	0.073	9.0	18.0	0.99680	3.36	0.57	9.5	7
8	7.5	0.500	0.36	6.1	0.071	17.0	102.0	0.99780	3.35	0.80	10.5	5
9	6.7	0.580	0.08	1.8	0.097	15.0	65.0	0.99590	3.28	0.54	9.2	5
10	7.5	0.500	0.36	6.1	0.071	17.0	102.0	0.99780	3.35	0.80	10.5	5
11	5.6	0.615	0.00	1.6	0.089	16.0	59.0	0.99430	3.58	0.52	9.9	5
12	7.8	0.610	0.29	1.6	0.114	9.0	29.0	0.99740	3.26	1.56	9.1	5
13	8.9	0.620	0.18	3.8	0.176	52.0	145.0	0.99860	3.16	0.88	9.2	5
14	8.9	0.620	0.19	3.9	0.170	51.0	148.0	0.99860	3.17	0.93	9.2	5
15	8.5	0.280	0.56	1.8	0.092	35.0	103.0	0.99690	3.30	0.75	10.5	7
16	8.1	0.560	0.28	1.7	0.368	16.0	56.0	0.99680	3.11	1.28	9.3	5
17	7.4	0.590	0.08	4.4	0.086	6.0	29.0	0.99740	3.38	0.50	9.0	4
18	7.9	0.320	0.51	1.8	0.341	17.0	56.0	0.99690	3.04	1.08	9.2	6
19	8.9	0.220	0.48	1.8	0.077	29.0	60.0	0.99680	3.39	0.53	9.4	6
20	7.6	0.390	0.31	2.3	0.082	23.0	71.0	0.99820	3.52	0.65	9.7	5
21	7.9	0.430	0.21	1.6	0.106	10.0	37.0	0.99660	3.17	0.91	9.5	5
22	8.5	0.490	0.11	2.3	0.084	9.0	67.0	0.99680	3.17	0.53	9.4	5
23	6.9	0.400	0.14	2.4	0.085	21.0	40.0	0.99680	3.43	0.63	9.7	6
24	6.3	0.390	0.16	1.4	0.080	11.0	23.0	0.99550	3.34	0.56	9.3	5
25	7.6	0.410	0.24	1.8	0.080	4.0	11.0	0.99620	3.28	0.59	9.5	5

Reading data

```
df = pd.read_csv('winequality-red.csv',
sep=';', header=None)

df
```

	0	1	2	3	4	5	6	7	8	9	10	11
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
5	7.4	0.660	0.00	1.8	0.075	13.0	40.0	0.99780	3.51	0.56	9.4	5
6	7.9	0.600	0.06	1.6	0.069	15.0	59.0	0.99640	3.30	0.46	9.4	5
7	7.3	0.650	0.00	1.2	0.065	15.0	21.0	0.99460	3.39	0.47	10.0	7
8	7.8	0.580	0.02	2.0	0.073	9.0	18.0	0.99680	3.36	0.57	9.5	7
9	7.5	0.500	0.36	6.1	0.071	17.0	102.0	0.99780	3.35	0.80	10.5	5
10	6.7	0.580	0.08	1.8	0.097	15.0	65.0	0.99590	3.28	0.54	9.2	5
11	7.5	0.500	0.36	6.1	0.071	17.0	102.0	0.99780	3.35	0.80	10.5	5
12	5.6	0.615	0.00	1.6	0.089	16.0	59.0	0.99430	3.58	0.52	9.9	5
13	7.8	0.610	0.29	1.6	0.114	9.0	29.0	0.99740	3.26	1.56	9.1	5
14	8.9	0.620	0.18	3.8	0.176	52.0	145.0	0.99860	3.16	0.88	9.2	5
15	8.9	0.620	0.19	3.9	0.170	51.0	148.0	0.99860	3.17	0.93	9.2	5
16	8.5	0.280	0.56	1.8	0.092	35.0	103.0	0.99690	3.30	0.75	10.5	7
17	8.1	0.560	0.28	1.7	0.368	16.0	56.0	0.99680	3.11	1.28	9.3	5
18	7.4	0.590	0.08	4.4	0.086	6.0	29.0	0.99740	3.38	0.50	9.0	4
19	7.9	0.320	0.51	1.8	0.341	17.0	56.0	0.99690	3.04	1.08	9.2	6
20	8.9	0.220	0.48	1.8	0.077	29.0	60.0	0.99680	3.39	0.53	9.4	6
21	7.6	0.390	0.31	2.3	0.082	23.0	71.0	0.99820	3.52	0.65	9.7	5
22	7.9	0.430	0.21	1.6	0.106	10.0	37.0	0.99660	3.17	0.91	9.5	5
23	8.5	0.490	0.11	2.3	0.084	9.0	67.0	0.99680	3.17	0.53	9.4	5
24	6.9	0.400	0.14	2.4	0.085	21.0	40.0	0.99680	3.43	0.63	9.7	6
25	6.3	0.390	0.16	1.4	0.080	11.0	23.0	0.99550	3.34	0.56	9.3	5

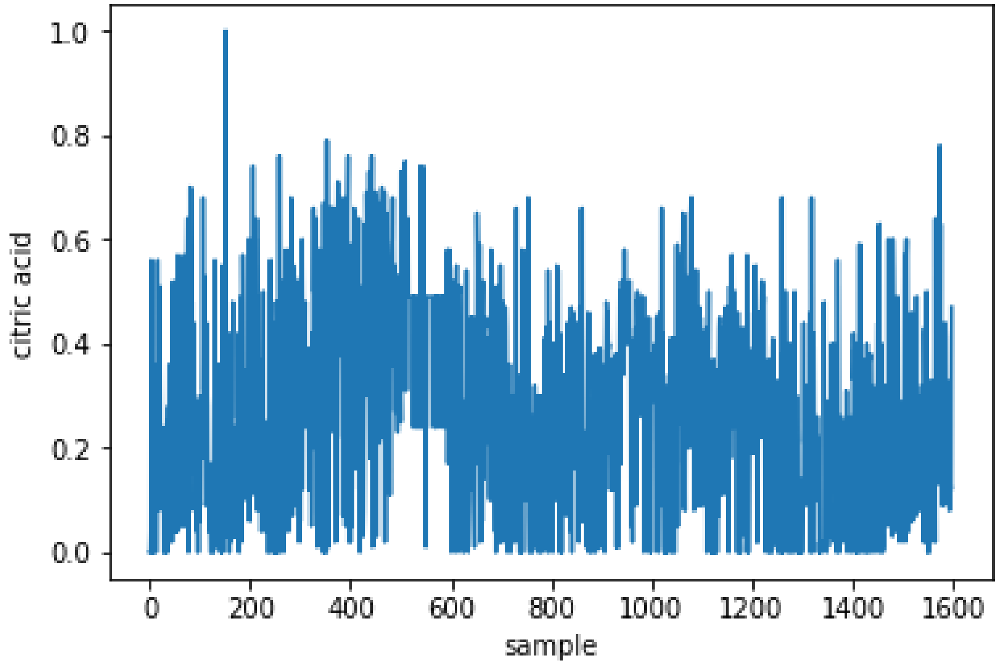
Reading data

```
df = pd.read_csv('winequality-red-header.csv', sep=';')
df
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
5	7.4	0.660	0.00	1.8	0.075	13.0	40.0	0.99780	3.51	0.56	9.4	5
6	7.9	0.600	0.06	1.6	0.069	15.0	59.0	0.99640	3.30	0.46	9.4	5
7	7.3	0.650	0.00	1.2	0.065	15.0	21.0	0.99460	3.39	0.47	10.0	7
8	7.8	0.580	0.02	2.0	0.073	9.0	18.0	0.99680	3.36	0.57	9.5	7
9	7.5	0.500	0.36	6.1	0.071	17.0	102.0	0.99780	3.35	0.80	10.5	5
10	6.7	0.580	0.08	1.8	0.097	15.0	65.0	0.99590	3.28	0.54	9.2	5
11	7.5	0.500	0.36	6.1	0.071	17.0	102.0	0.99780	3.35	0.80	10.5	5
12	5.6	0.615	0.00	1.6	0.089	16.0	59.0	0.99430	3.58	0.52	9.9	5
13	7.8	0.610	0.29	1.6	0.114	9.0	29.0	0.99740	3.26	1.56	9.1	5
14	8.9	0.620	0.18	3.8	0.176	52.0	145.0	0.99860	3.16	0.88	9.2	5
15	8.9	0.620	0.19	3.9	0.170	51.0	148.0	0.99860	3.17	0.93	9.2	5
16	8.5	0.280	0.56	1.8	0.092	35.0	103.0	0.99690	3.30	0.75	10.5	7
17	8.1	0.560	0.28	1.7	0.368	16.0	56.0	0.99680	3.11	1.28	9.3	5
18	7.4	0.590	0.08	4.4	0.086	6.0	29.0	0.99740	3.38	0.50	9.0	4
19	7.9	0.320	0.51	1.8	0.341	17.0	56.0	0.99690	3.04	1.08	9.2	6
20	8.9	0.220	0.48	1.8	0.077	29.0	60.0	0.99680	3.39	0.53	9.4	6
21	7.6	0.390	0.31	2.3	0.082	23.0	71.0	0.99820	3.52	0.65	9.7	5
22	7.9	0.430	0.21	1.6	0.106	10.0	37.0	0.99660	3.17	0.91	9.5	5
23	8.5	0.490	0.11	2.3	0.084	9.0	67.0	0.99680	3.17	0.53	9.4	5
24	6.9	0.400	0.14	2.4	0.085	21.0	40.0	0.99680	3.43	0.63	9.7	6
25	6.3	0.390	0.16	1.4	0.080	11.0	23.0	0.99550	3.34	0.56	9.3	5
26	7.6	0.410	0.24	1.8	0.080	4.0	11.0	0.99620	3.28	0.59	9.5	5

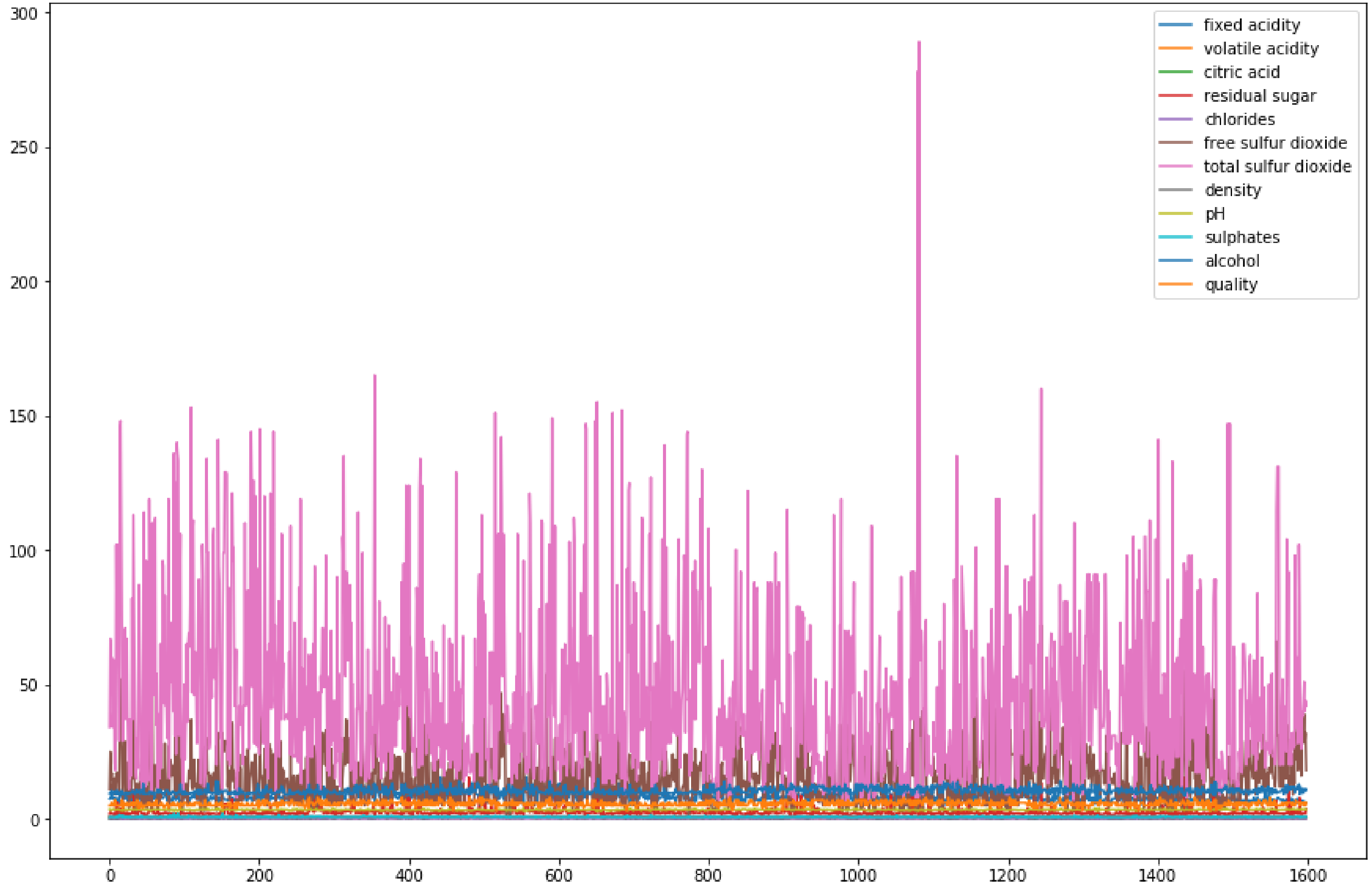
Plotting data

```
df['citric acid'].plot()
plt.xlabel('sample')
plt.ylabel('citric acid')
```



Plotting data

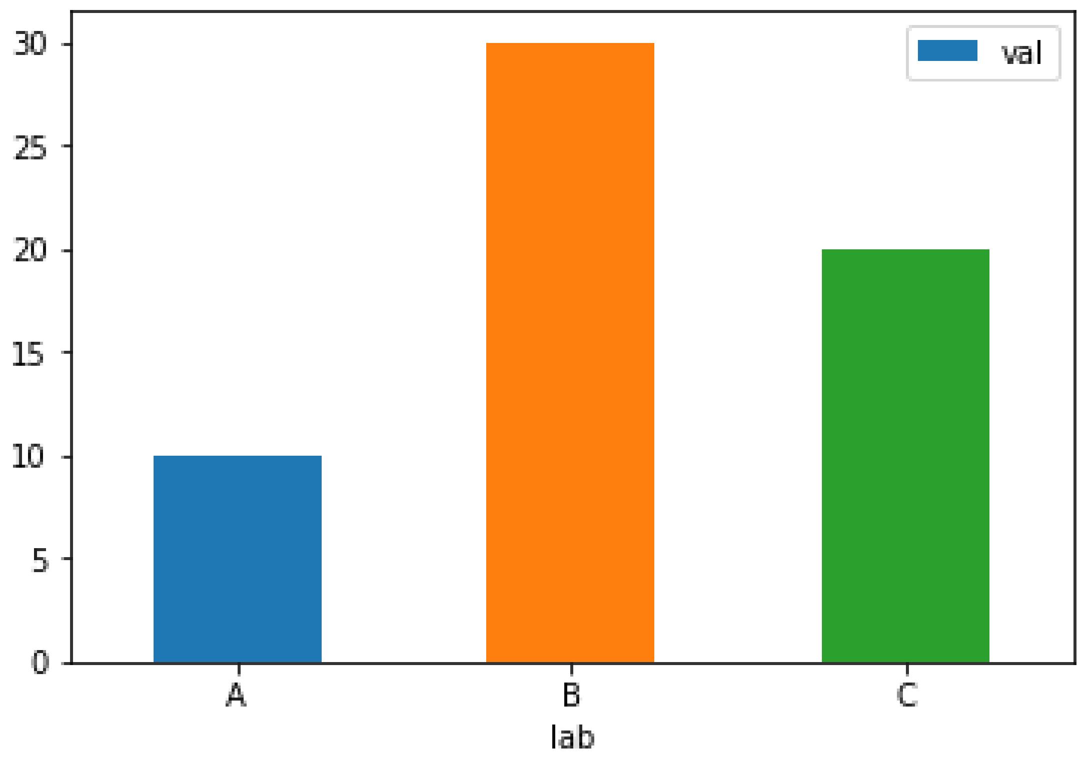
```
df.plot(figsize=(15, 10))
```



Plotting data

```
df = pd.DataFrame(
{
 'lab':['A', 'B', 'C’],
 'val':[10, 30, 20]}
)

ax = df.plot.bar(
 x='lab’,
 y='val’,
 rot=0
)
```



Plotting data

More:

https://pandas.pydata.org/pandas-docs/version/0.23/generated/pandas.DataFrame.plot.bar.html