

# Matplotlib

## A tutorial

Devert Alexandre  
School of Software Engineering of USTC



# Table of Contents

- ① First steps
- ② Curve plots
- ③ Scatter plots
- ④ Boxplots
- ⑤ Histograms
- ⑥ Usage example



# Curve plot

Let's plot a curve

---

```
import math
import matplotlib.pyplot as plt

# Generate a sinusoid
nbSamples = 256
xRange = (-math.pi, math.pi)

x, y = [], []
for n in xrange(nbSamples):
    k = (n + 0.5) / nbSamples
    x.append(xRange[0] + (xRange[1] - xRange[0]) * k)
    y.append(math.sin(x[-1]))

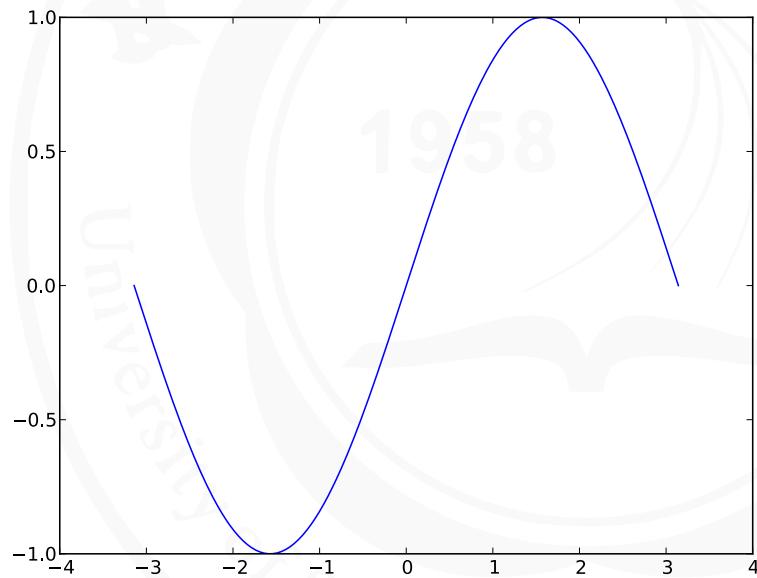
# Plot the sinusoid
plt.plot(x, y)
plt.show()
```

---



# Curve plot

This will show you something like this



# numpy

*matplotlib* can work with *numpy* arrays

---

```
import math
import numpy
import matplotlib.pyplot as plt

# Generate a sinusoid
nbSamples = 256
xRange = (-math.pi, math.pi)

x, y = numpy.zeros(nbSamples), numpy.zeros(nbSamples)
for n in xrange(nbSamples):
    k = (n + 0.5) / nbSamples
    x[n] = xRange[0] + (xRange[1] - xRange[0]) * k
    y[n] = math.sin(x[n])

# Plot the sinusoid
plt.plot(x, y)
plt.show()
```

---



*numpy* provides a lot of function and is efficient

## numpy

- `zeros` build arrays filled of 0
- `linspace` build arrays filled with an arithmetic sequence

---

```
import math
import numpy
import matplotlib.pyplot as plt

# Generate a sinusoid
x = numpy.linspace(-math.pi, math.pi, num=256)
y = numpy.zeros(nbSamples)
for n in xrange(nbSamples):
    y[n] = math.sin(x[n])

# Plot the sinusoid
plt.plot(x, y)
plt.show()
```

---



# numpy

*numpy functions can work on entire arrays*

---

```
import math
import numpy
import matplotlib.pyplot as plt

# Generate a sinusoid
x = numpy.linspace(-math.pi, math.pi, num=256)
y = numpy.sin(x)

# Plot the sinusoid
plt.plot(x, y)
plt.show()
```

---



# PDF output

Exporting to a PDF file is just one change

---

```
import math
import numpy
import matplotlib.pyplot as plt

# Generate a sinusoid
x = numpy.linspace(-math.pi, math.pi, num=256)
y = numpy.sin(x)

# Plot the sinusoid
plt.plot(x, y)
plt.savefig('sin-plot.pdf', transparent=True)
```

---



# Table of Contents

- ① First steps
- ② Curve plots
- ③ Scatter plots
- ④ Boxplots
- ⑤ Histograms
- ⑥ Usage example



# Multiple curves

It's often convenient to show several curves in one figure

---

```
import math
import numpy
import matplotlib.pyplot as plt

# Generate a sinusoid and a cosinoid
x = numpy.linspace(-math.pi, math.pi, num=256)
y = numpy.sin(x)
z = numpy.cos(x)

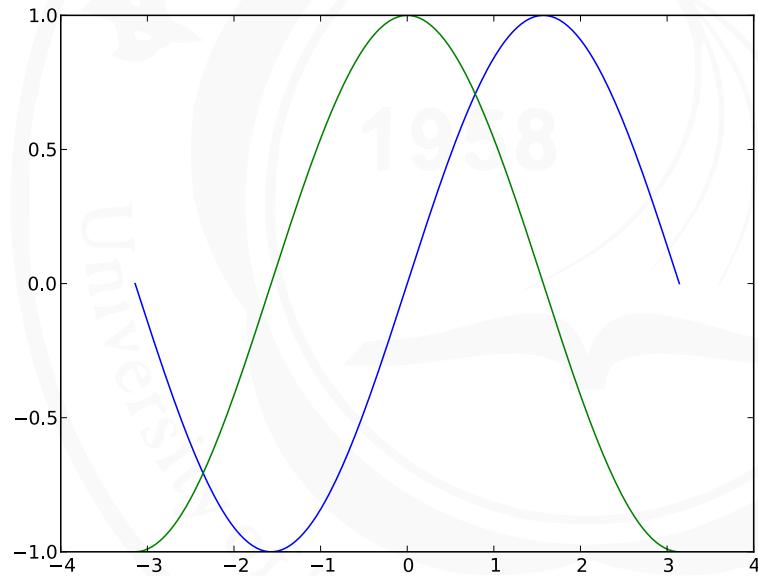
# Plot the curves
plt.plot(x, y)
plt.plot(x, z)
plt.show()
```

---



# Multiple curves

It's often convenient to show several curves in one figure



# Custom colors

Changing colors can help to make nice documents

---

```
import math
import numpy
import matplotlib.pyplot as plt

# Generate a sinusoid and a cosinoid
x = numpy.linspace(-math.pi, math.pi, num=256)
y = numpy.sin(x)
z = numpy.cos(x)

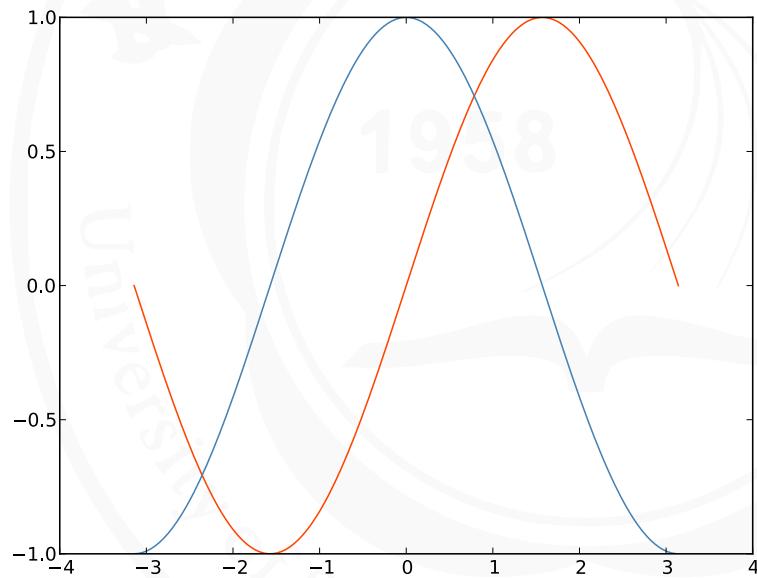
# Plot the curves
plt.plot(x, y, c="#FF4500")
plt.plot(x, z, c="#4682B4")
plt.show()
```

---



# Custom colors

Changing colors can help to make nice documents



# Line thickness

Line thickness can be changed as well

---

```
import math
import numpy
import matplotlib.pyplot as plt

# Generate a sinusoid and a cosinoid
x = numpy.linspace(-math.pi, math.pi, num=256)
y = numpy.sin(x)
z = numpy.cos(x)

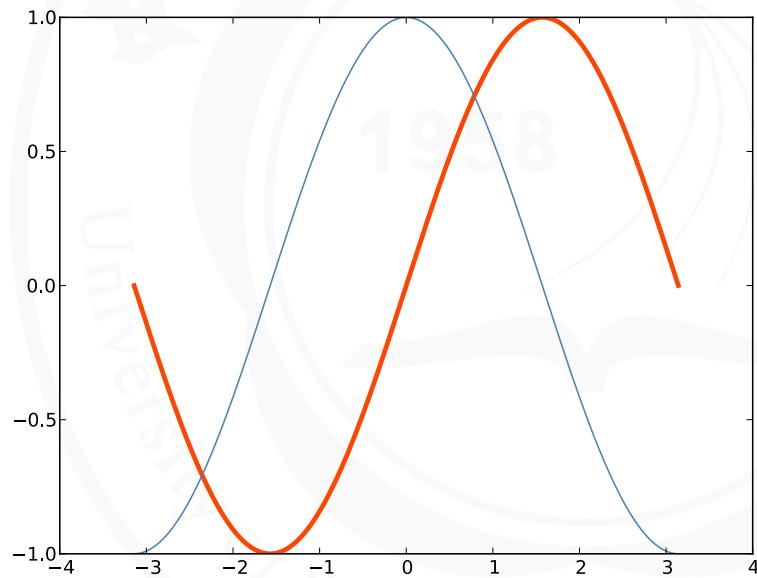
# Plot the curves
plt.plot(x, y, linewidth=3, c='FF4500')
plt.plot(x, z, c='#4682B4')
plt.show()
```

---



# Line thickness

Line thickness can be changed as well



# Line patterns

For printed document, colors can be replaced by line patterns

---

```
import math
import numpy
import matplotlib.pyplot as plt

# Linestyles can be '--', '---', '-.', ':'

# Generate a sinusoid and a cosinoid
x = numpy.linspace(-math.pi, math.pi, num=256)
y = numpy.sin(x)
z = numpy.cos(x)

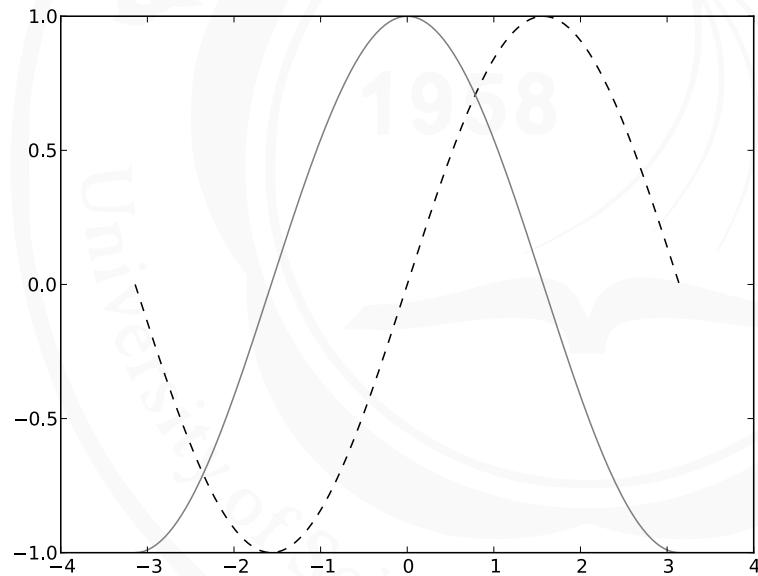
# Plot the curves
plt.plot(x, y, linestyle='--', c='#000000')
plt.plot(x, z, c='#808080')
plt.show()
```

---



## Line patterns

For printed document, colors can be replaced by line patterns



# Markers

It sometime relevant to show the data points

---

```
import math
import numpy
import matplotlib.pyplot as plt

# Markers can be '.', ',', 'o', '1' and more

# Generate a sinusoid and a cosinoid
x = numpy.linspace(-math.pi, math.pi, num=64)
y = numpy.sin(x)
z = numpy.cos(x)

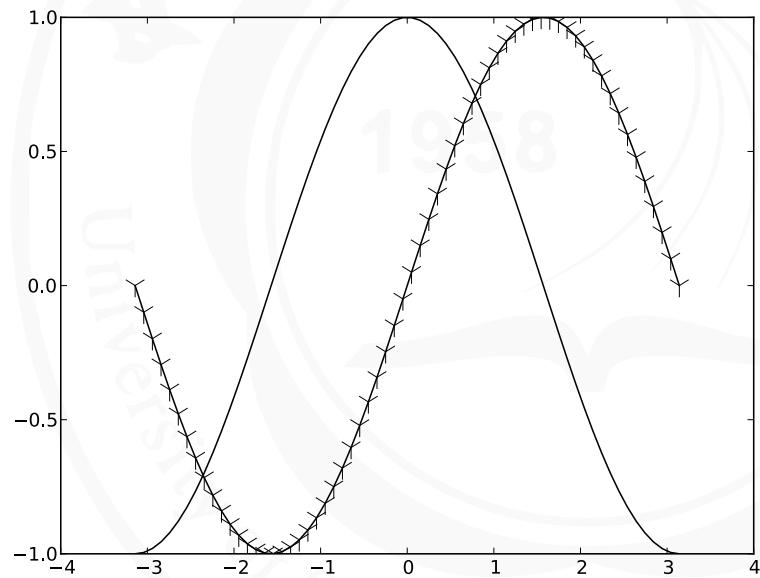
# Plot the curves
plt.plot(x, y, marker='1', markersize=15, c='#000000')
plt.plot(x, z, c='#000000')
plt.show()
```

---



# Markers

It sometime relevant to show the data points



# Legend

A legend can help to make self-explanatory figures

---

```
import math
import numpy
import matplotlib.pyplot as plt

# legend location can be 'best', 'center', 'left', 'right', etc.

# Generate a sinusoid and a cosinoid
x = numpy.linspace(-math.pi, math.pi, num=256)
y = numpy.sin(x)
z = numpy.cos(x)

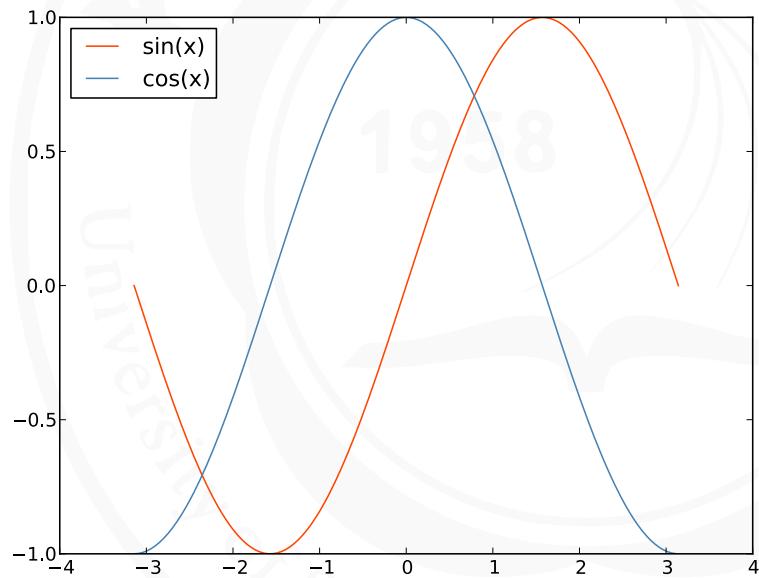
# Plot the curves
plt.plot(x, y, c="#FF4500", label='sin(x)')
plt.plot(x, z, c="#4682B4", label='cos(x)')
plt.legend(loc='best')
plt.show()
```

---



# Legend

A legend can help to make self-explanatory figures



# Custom axis scale

Changing the axis scale can improve readability

---

```
import math
import numpy
import matplotlib.pyplot as plt

# legend location can be 'best', 'center', 'left', 'right', etc.

# Generate a sinusoid and a cosinoid
x = numpy.linspace(-math.pi, math.pi, num=256)
y = numpy.sin(x)
z = numpy.cos(x)

# Axis setup
fig = plt.figure()
axis = fig.add_subplot(111)

axis.set_ylimits(-0.5 * math.pi, 0.5 * math.pi)

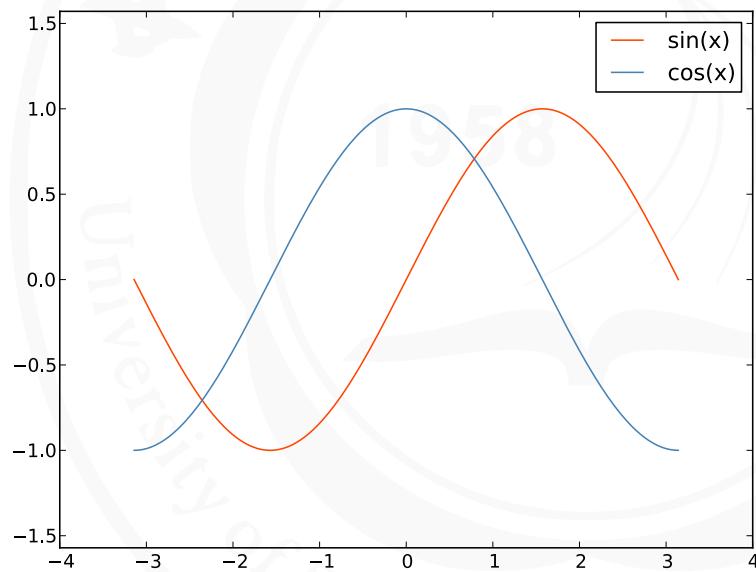
# Plot the curves
plt.plot(x, y, c='FF4500', label='sin(x)')
plt.plot(x, z, c='4682B4', label='cos(x)')
plt.legend(loc='best')
plt.show()
```

---



## Custom axis scale

Changing the axis scale can improve readability



# Grid

Same goes for a grid, can be helpful

---

```
import math
import numpy
import matplotlib.pyplot as plt

# legend location can be 'best', 'center', 'left', 'right', etc.

# Generate a sinusoid and a cosinoid
x = numpy.linspace(-math.pi, math.pi, num=256)
y = numpy.sin(x)
z = numpy.cos(x)

# Axis setup
fig = plt.figure()
axis = fig.add_subplot(111)

axis.set_ylim(-0.5 * math.pi, 0.5 * math.pi)
axis.grid(True)

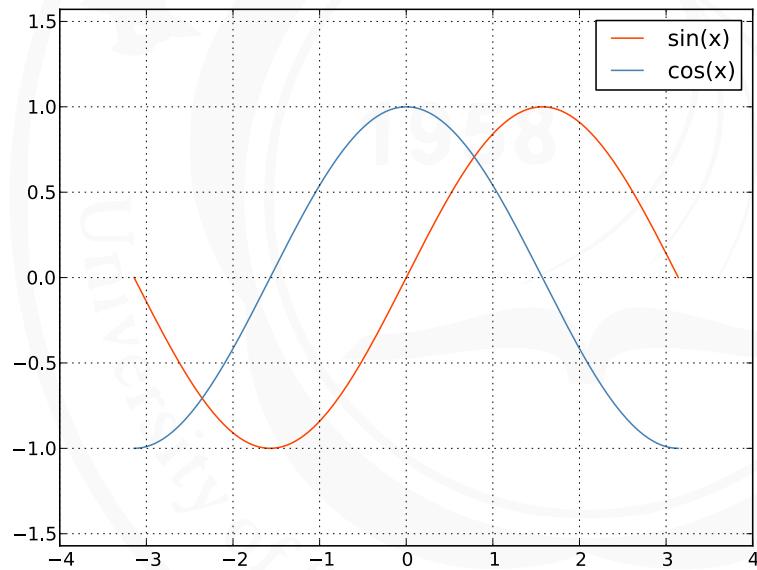
# Plot the curves
plt.plot(x, y, c='FF4500', label='sin(x)')
plt.plot(x, z, c='#4682B4', label='cos(x)')
plt.legend(loc='best')
plt.show()
```

---



# Grid

Same goes for a grid, can be helpful



# Error bars

Your data might come with a known measure error

---

```
import math
import numpy
import numpy.random
import matplotlib.pyplot as plt

# Generate a noisy sinusoid
x = numpy.linspace(-math.pi, math.pi, num=48)
y = numpy.sin(x + 0.05 * numpy.random.standard_normal(len(x)))
yerror = 0.1 * numpy.random.standard_normal(len(x))

# Axis setup
fig = plt.figure()
axis = fig.add_subplot(111)

axis.set_ylim(-0.5 * math.pi, 0.5 * math.pi)

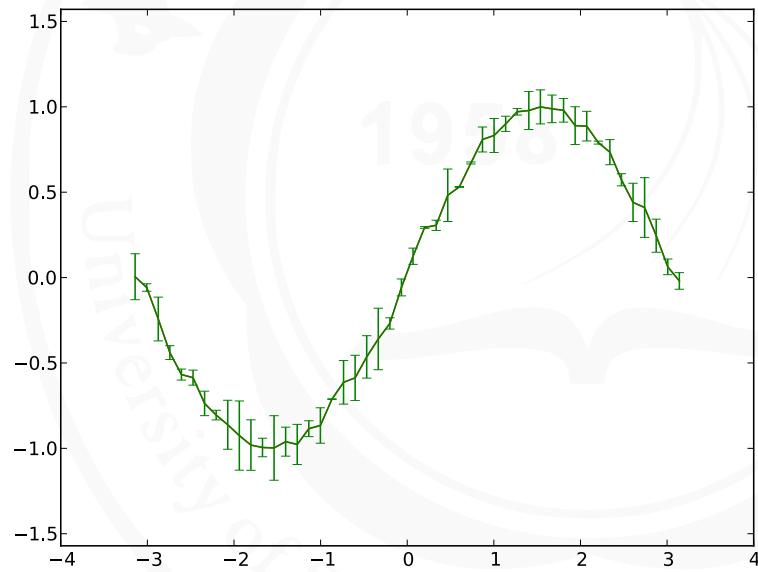
# Plot the curves
plt.plot(x, y, c='FF4500')
plt.errorbar(x, y, yerr=yerror)
plt.show()
```

---



# Error bars

Your data might come with a known measure error



# Table of Contents

- ① First steps
- ② Curve plots
- ③ Scatter plots
- ④ Boxplots
- ⑤ Histograms
- ⑥ Usage example



# Scatter plot

A scatter plot just shows one point for each dataset entry

---

```
import numpy
import numpy.random
import matplotlib.pyplot as plt

# Generate a 2d normal distribution
nbPoints = 100
x = numpy.random.standard_normal(nbPoints)
y = numpy.random.standard_normal(nbPoints)

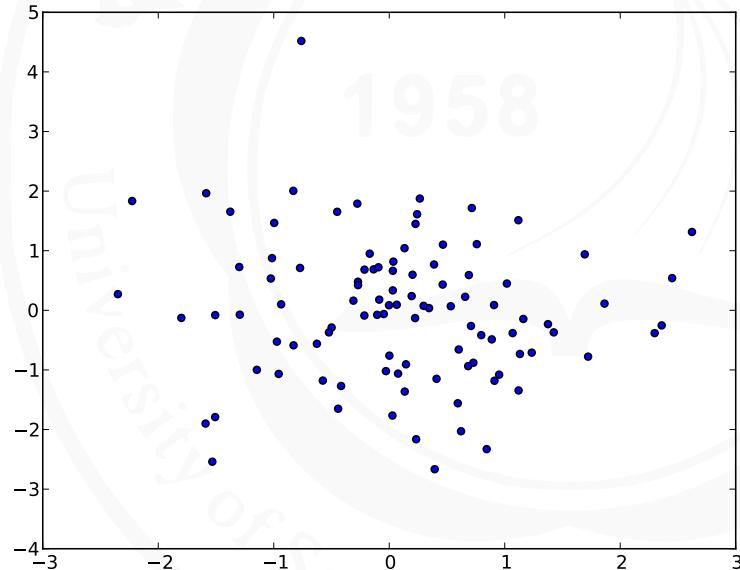
# Plot the points
plt.scatter(x, y)
plt.show()
```

---



# Scatter plot

A scatter plot just shows one point for each dataset entry



# Aspect ratio

If can be very important to have the same scale on both axis

---

```
import numpy
import numpy.random
import matplotlib.pyplot as plt

# Generate a 2d normal distribution
nbPoints = 100
x = numpy.random.standard_normal(nbPoints)
y = 0.1 * numpy.random.standard_normal(nbPoints)

# Axis setup
fig = plt.figure()
axis = fig.add_subplot(111, aspect='equal')

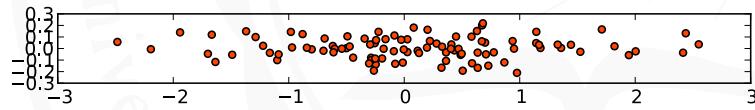
# Plot the points
plt.scatter(x, y, c='FF4500')
plt.show()
```

---



## Aspect ratio

If can be very important to have the same scale on both axis



# Aspect ratio

Alternative way to keep the same scale on both axis

---

```
import numpy
import numpy.random
import matplotlib.pyplot as plt

# Generate a 2d normal distribution
nbPoints = 100
x = numpy.random.standard_normal(nbPoints)
y = 0.1 * numpy.random.standard_normal(nbPoints)

# Axis setup
fig = plt.figure()
axis = fig.add_subplot(111)

cmin, cmax = min(min(x), min(y)), max(max(x), max(y))
cmin -= 0.05 * (cmax - cmin)
cmax += 0.05 * (cmax - cmin)

axis.set_xlim(cmin, cmax)
axis.set_ylim(cmin, cmax)

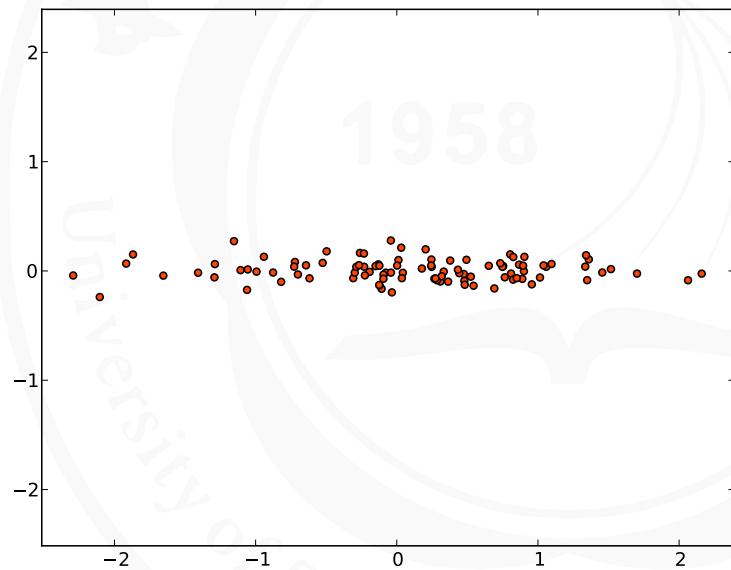
# Plot the points
plt.scatter(x, y, c='#FF4500')
plt.show()
```

---



# Aspect ratio

Alternative way to keep the same scale on both axis



# Multiple scatter plots

As for curve, you can show 2 datasets on one figure

---

```
import numpy
import numpy.random
import matplotlib.pyplot as plt

colors = ('#FF4500', '#3CB371', '#4682B4', '#DB7093', '#FFD700')

# Generate a 2d normal distribution
nbPoints = 100

x, y = [], []
x += [numpy.random.standard_normal(nbPoints)]
y += [0.25 * numpy.random.standard_normal(nbPoints)]

x += [0.5 * numpy.random.standard_normal(nbPoints) + 3.0]
y += [2 * numpy.random.standard_normal(nbPoints) + 2.0]

# Axis setup
fig = plt.figure()
axis = fig.add_subplot(111, aspect='equal')

# Plot the points
for i in xrange(len(x)):
    plt.scatter(x[i], y[i], c=colors[i % len(colors)])

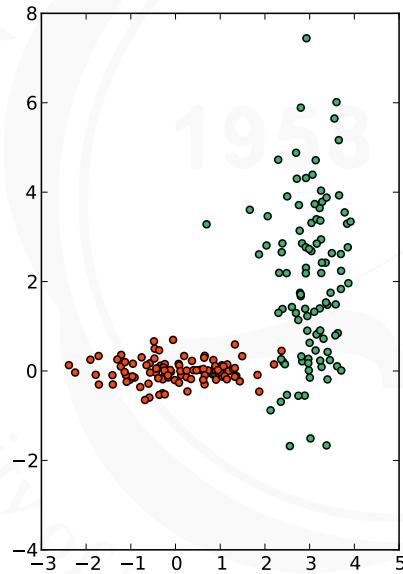
plt.show()
```

---



## Multiple scatter plots

As for curve, you can show 2 datasets on one figure



# Showing centers

It can help to see the centers or the median points

```
import numpy
import numpy.random
import matplotlib.pyplot as plt

colors = ('#FF4500', '#3CB371', '#4682B4', '#DB7093', '#FFD700')

# Generate a 2d normal distribution
nbPoints = 100

x, y = [], []

x += [numpy.random.standard_normal(nbPoints)]
y += [0.25 * numpy.random.standard_normal(nbPoints)]

x += [0.5 * numpy.random.standard_normal(nbPoints) + 3.0]
y += [2 * numpy.random.standard_normal(nbPoints) + 2.0]

# Axis setup
fig = plt.figure()
axis = fig.add_subplot(111, aspect='equal')

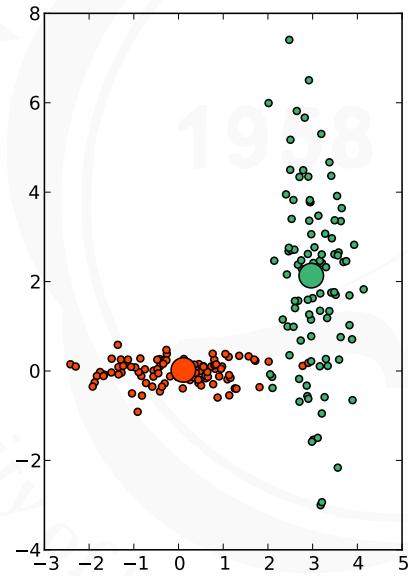
# Plot the points
for i in xrange(len(x)):
    col = colors[i % len(colors)]
    plt.scatter(x[i], y[i], c=col)
    plt.scatter([numpy.median(x[i])], [numpy.median(y[i])], c=col, s=250)

plt.show()
```



## Showing centers

It can help to see the centers or the median points



# Marker styles

You can use different markers styles

```
import numpy
import numpy.random
import matplotlib.pyplot as plt

markers = ('+', '^', '.')

# Generate a 2d normal distribution
nbPoints = 100

x, y = [], []

x += [numpy.random.standard_normal(nbPoints)]
y += [0.25 * numpy.random.standard_normal(nbPoints)]

x += [0.5 * numpy.random.standard_normal(nbPoints) + 3.0]
y += [2 * numpy.random.standard_normal(nbPoints) + 2.0]

# Axis setup
fig = plt.figure()
axis = fig.add_subplot(111, aspect='equal')

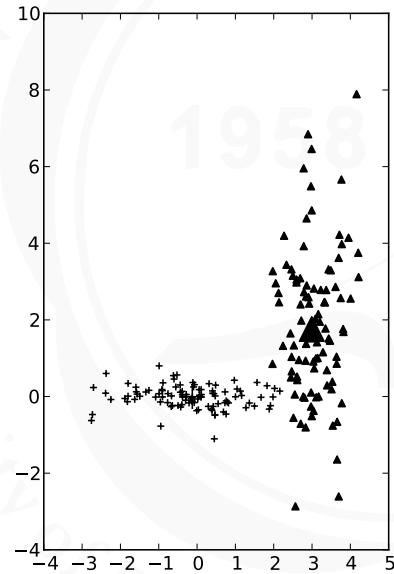
# Plot the points
for i in xrange(len(x)):
    m = markers[i % len(markers)]
    plt.scatter(x[i], y[i], marker=m, c='#000000')
    plt.scatter([numpy.median(x[i])], [numpy.median(y[i])], marker=m, s=250, c='#000000')

plt.show()
```



# Marker styles

You can use different markers styles



# Table of Contents

- ① First steps
- ② Curve plots
- ③ Scatter plots
- ④ Boxplots
- ⑤ Histograms
- ⑥ Usage example



# Boxplots

Let's do a simple boxplot

---

```
import numpy
import numpy.random
import matplotlib.pyplot as plt

# Generate normal distribution data
x = numpy.random.standard_normal(256)

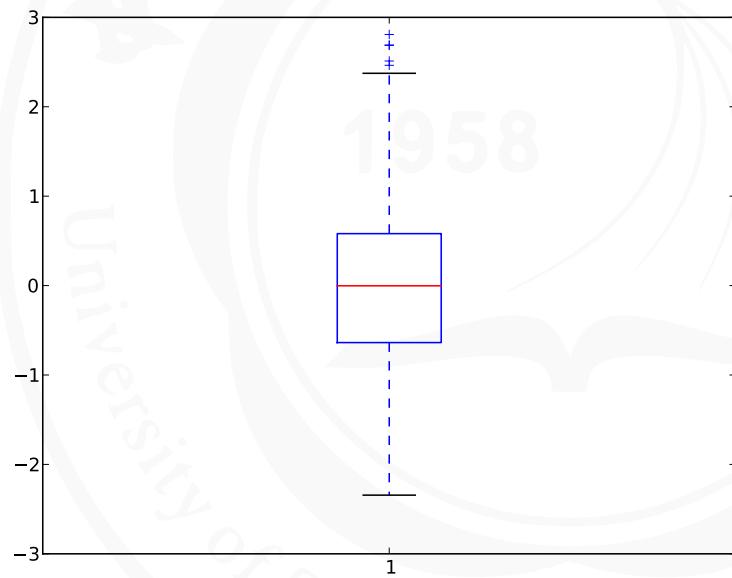
# Show a boxplot of the data
plt.boxplot(x)
plt.show()
```

---



# Boxplots

Let's do a simple boxplot



# Boxplots

You might want to show the original data in the same time

---

```
import numpy
import numpy.random
import matplotlib.pyplot as plt

# Generate normal distribution data
x = numpy.random.standard_normal(256)

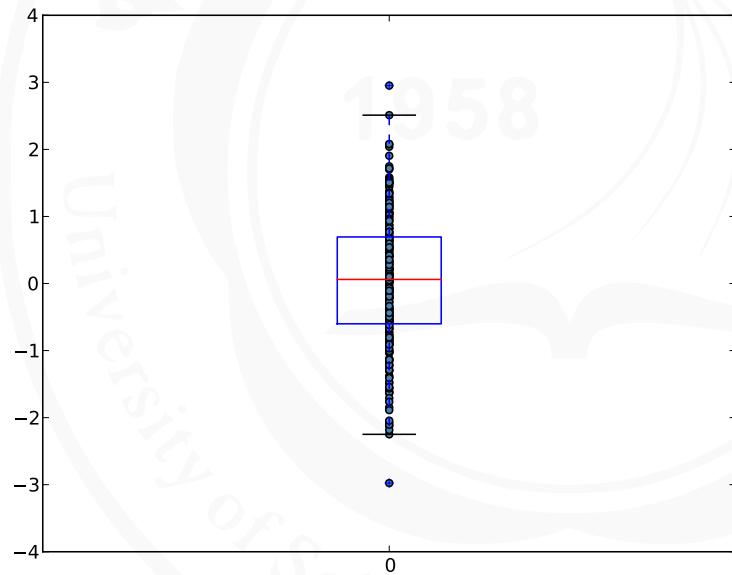
# Show a boxplot of the data
plt.scatter([0] * len(x), x, c='#4682B4')
plt.boxplot(x, positions=[0])
plt.show()
```

---



## Boxplots

You might want to show the original data in the same time



# Multiple boxplots

Boxplots are often used to show side by side various distributions

---

```
import numpy
import numpy.random
import matplotlib.pyplot as plt

# Generate normal distribution data
data = []
for i in xrange(5):
    mu = 10 * numpy.random.random_sample()
    sigma = 2 * numpy.random.random_sample() + 0.1
    data.append(numpy.random.normal(mu, sigma, 256))

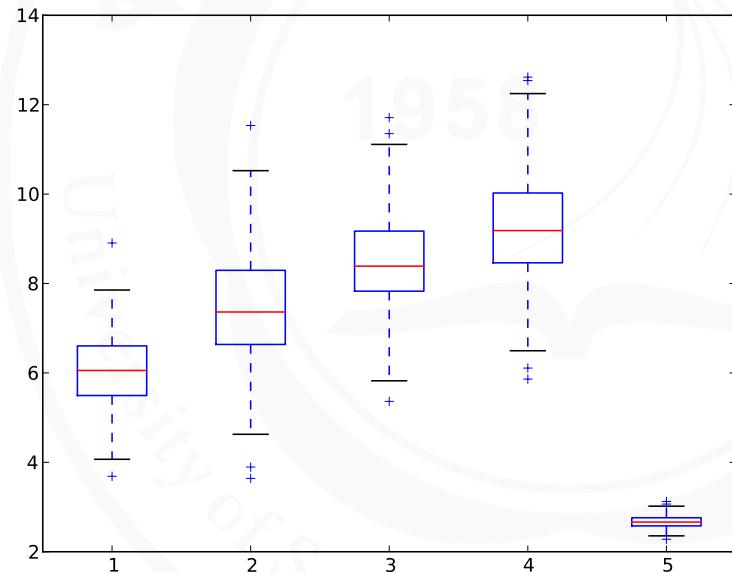
# Show a boxplot of the data
plt.boxplot(data)
plt.show()
```

---



## Multiple boxplots

Boxplots are often used to show side by side various distributions



# Orientation

Changing the orientation is easily done

---

```
import numpy
import numpy.random
import matplotlib.pyplot as plt

# Generate normal distribution data
data = []
for i in xrange(5):
    mu = 10 * numpy.random.random_sample()
    sigma = 2 * numpy.random.random_sample() + 0.1
    data.append(numpy.random.normal(mu, sigma, 256))

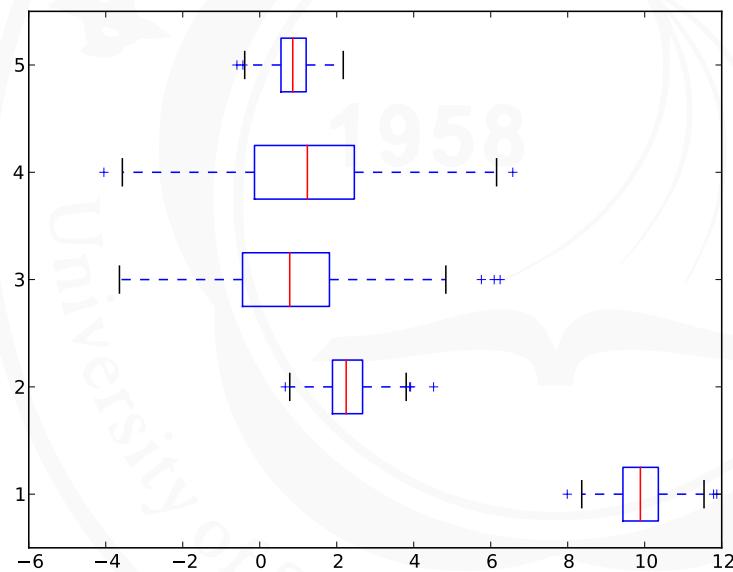
# Show a boxplot of the data
plt.boxplot(data, vert=0)
plt.show()
```

---



# Orientation

Changing the orientation is easily done



# Legend

Good graphics have a proper legend

---

```
import numpy
import numpy.random
import matplotlib.pyplot as plt

# Generate normal distribution data
labels = ['mercury', 'lead', 'lithium', 'tungstene', 'cadmium']
data = []
for i in xrange(len(labels)):
    mu = 10 * numpy.random.random_sample() + 100
    sigma = 2 * numpy.random.random_sample() + 0.1
    data.append(numpy.random.normal(mu, sigma, 256))

# Axis setup
fig = plt.figure()
axis = fig.add_subplot(111)

axis.set_title('Alien_nodules_composition')
xtickNames = plt.setp(axis, xticklabels=labels)
axis.set_ylabel('concentration_(ppm)')

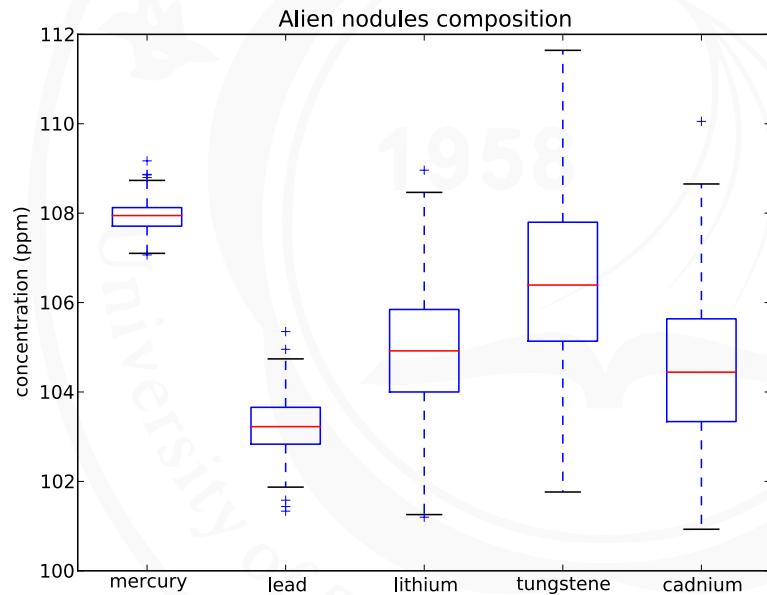
# Show a boxplot of the data
plt.boxplot(data)
plt.show()
```

---



# Legend

Good graphics have a proper legend



# Table of Contents

- ① First steps
- ② Curve plots
- ③ Scatter plots
- ④ Boxplots
- ⑤ Histograms
- ⑥ Usage example



# Histograms

Histogram are convenient to sum-up results

---

```
import numpy
import numpy.random
import matplotlib.pyplot as plt

# Some data
data = numpy.abs(numpy.random.standard_normal(30))

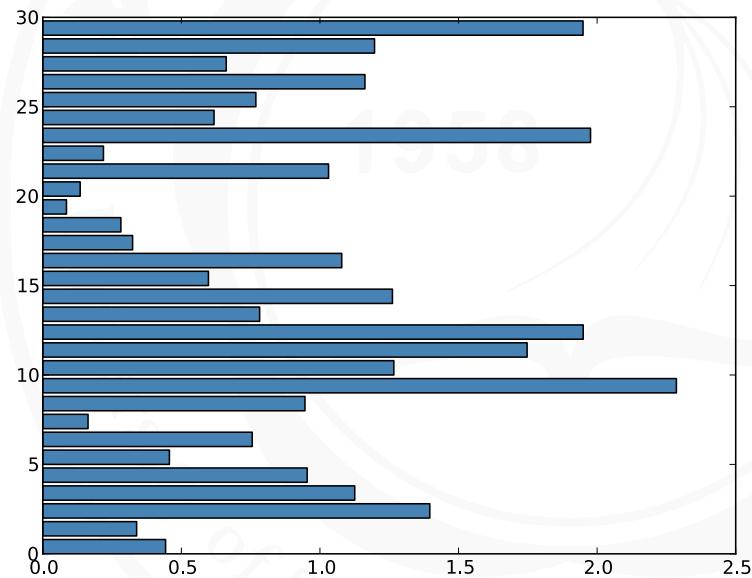
# Show an histogram
plt.barh(range(len(data)), data, color='#4682B4')
plt.show()
```

---



# Histograms

Histogram are convenient to sum-up results



# Histograms

A variant to show 2 quantities per item on 1 figure

---

```
import numpy
import numpy.random
import matplotlib.pyplot as plt

# Some data
data = [[], []]
for i in xrange(len(data)):
    data[i] = numpy.abs(numpy.random.standard_normal(30))

# Show an histogram
labels = range(len(data[0]))

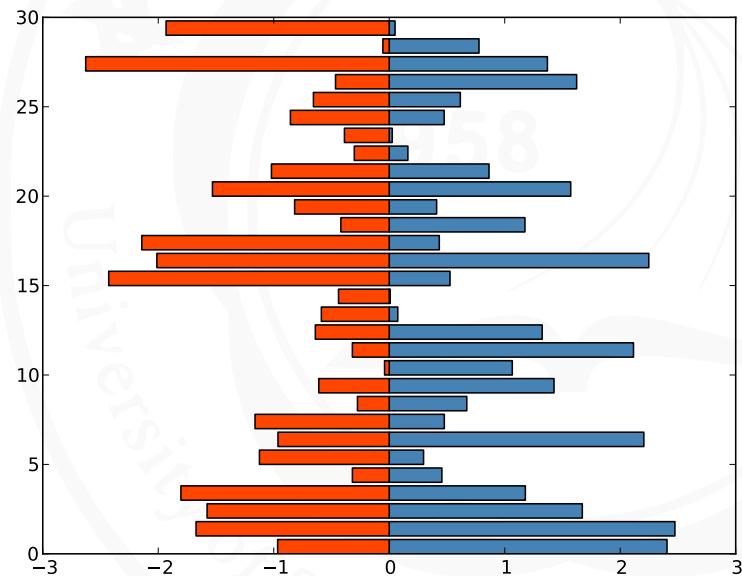
plt.bart(labels, data[0], color='#4682B4')
plt.bart(labels, -1 * data[1], color='FF4500')
plt.show()
```

---



# Histograms

A variant to show 2 quantities per item on 1 figure



# Labels

Very often, we need to name items on an histogram

---

```
import numpy
import numpy.random
import matplotlib.pyplot as plt

# Some data
names = [ 'Wang_Bu' , 'Cheng_Cao' , 'Zhang_Xue_Li' , 'Tang_Wei' , 'Sun_Wu_Kong' ]
marks = 7 * numpy.random.random_sample(len(names)) + 3

# Axis setup
fig = plt.figure()
axis = fig.add_subplot(111)

axis.set_xlim(0, 10)
plt.yticks(numpy.arange(len(marks))+0.5, names)

axis.set_title('Data-mining_marks')
axis.set_xlabel('mark')

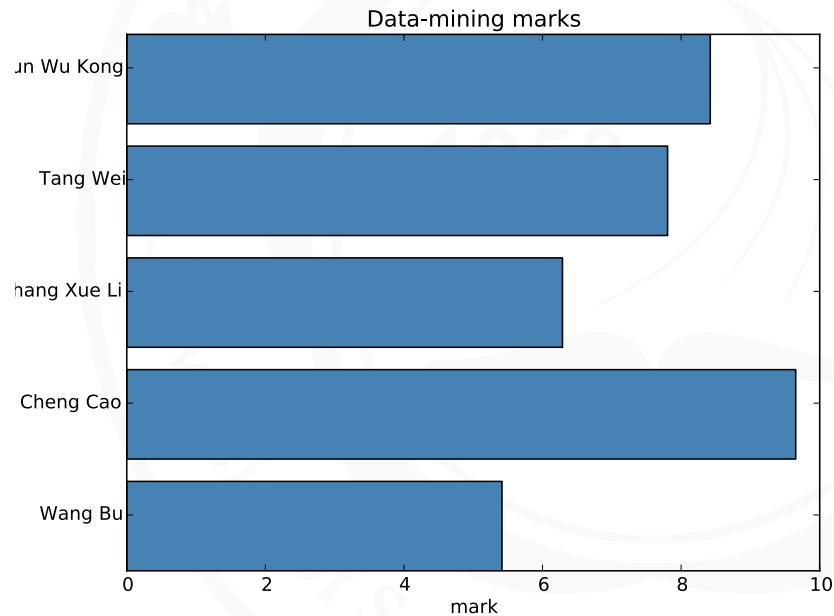
# Show an histogram
plt.bach(range(len(marks)), marks, color='#4682B4')
plt.show()
```

---



# Labels

Very often, we need to name items on an histogram



# Error bars

Error bars, to indicate the accuracy of values

---

```
import numpy
import numpy.random
import matplotlib.pyplot as plt

# Some data
names = ['6809', '6502', '8086', 'Z80', 'RCA1802']
speed = 70 * numpy.random.random_sample(len(names)) + 30
error = 9 * numpy.random.random_sample(len(names)) + 1

# Axis setup
fig = plt.figure()
axis = fig.add_subplot(111)

plt.yticks(numpy.arange(len(names))+0.5, names)

axis.set_title('8_bits_CPU_benchmark--string-test')
axis.set_xlabel('score')

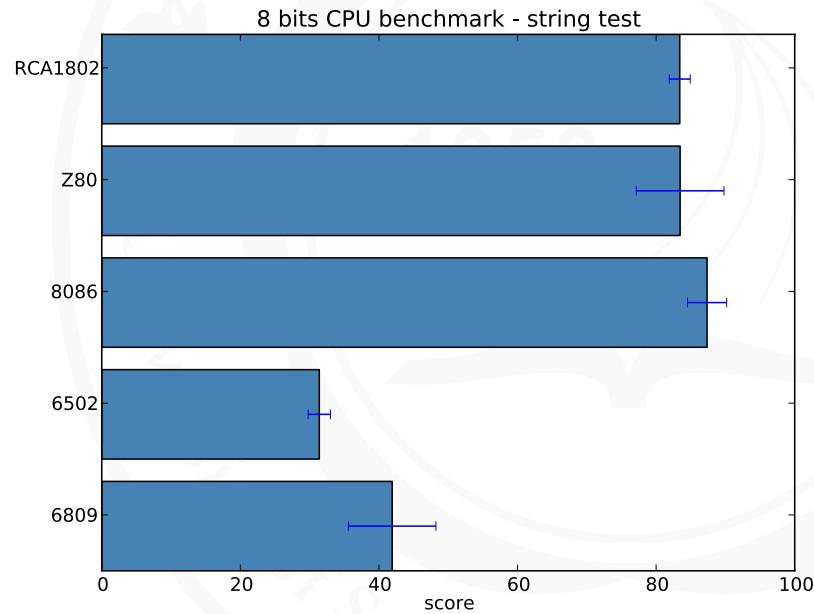
# Show an histogram
plt.barh(range(len(names)), speed, xerr=error, color='#4682B4')
plt.show()
```

---



## Error bars

Error bars, to indicate the accuracy of values



# Table of Contents

- ① First steps
- ② Curve plots
- ③ Scatter plots
- ④ Boxplots
- ⑤ Histograms
- ⑥ Usage example



# Old Faithful

Let's display some real data: *Old Faithful* geyser



# Old Faithful

This way works, but good example of half-done job

---

```
import numpy
import matplotlib.pyplot as plt

# Read the data
data = numpy.loadtxt('./datasets/geyser.dat')

# Axis setup
fig = plt.figure()
axis = fig.add_subplot(111, aspect='equal')

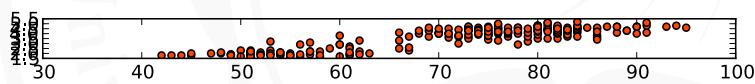
# Plot the points
plt.scatter(data[:,0], data[:,1], c='#FF4500')
plt.show()
```

---



# Old Faithful

This way works, but good example of half-done job



# Old Faithful

Let's make a more readable figure

---

```
import numpy
import matplotlib.pyplot as plt

# Read the data
data = numpy.loadtxt('./datasets/geyser.dat')

# Axis setup
fig = plt.figure()
axis = fig.add_subplot(111)

axis.set_title('Old_Faithful_geyser_dataset')
axis.set_xlabel('waiting_time_(mn.)')
axis.set_ylabel('eruption_duration_(mn.)')

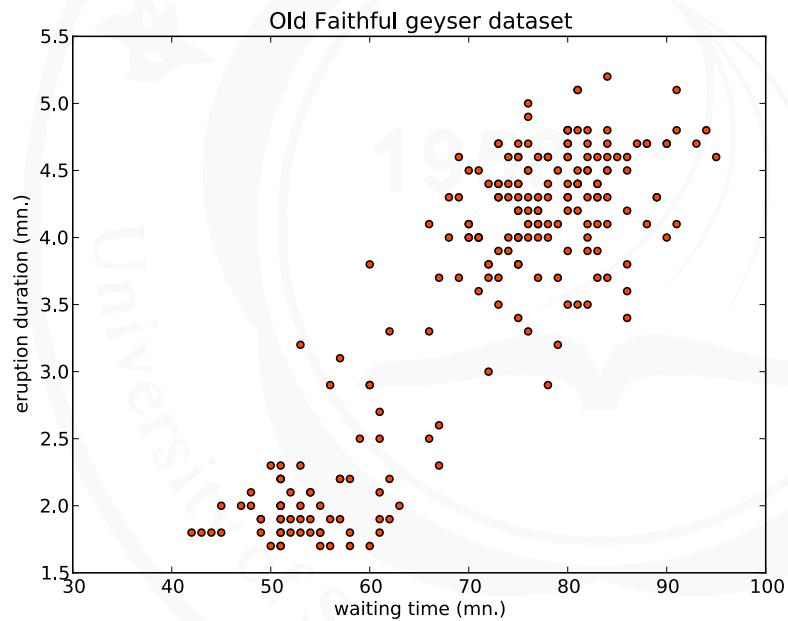
# Plot the points
plt.scatter(data[:,0], data[:,1], c='FF4500')
plt.show()
```

---



# Old Faithful

Let's make a more readable figure



# Mercury & fishes

Let's display more complex data: fishes and mercury poisoning



# Mercury & fishes

## A first try

---

```
import numpy
import matplotlib.pyplot as plt

# Read the data
data = numpy.loadtxt('./datasets/fish.dat')

# Axis setup
fig = plt.figure()
axis = fig.add_subplot(111)

axis.set_title('North_Carolina_fishes_mercury_concentration')
axis.set_xlabel('weight_(g.)')
axis.set_ylabel('mercury_concentration_(ppm)')

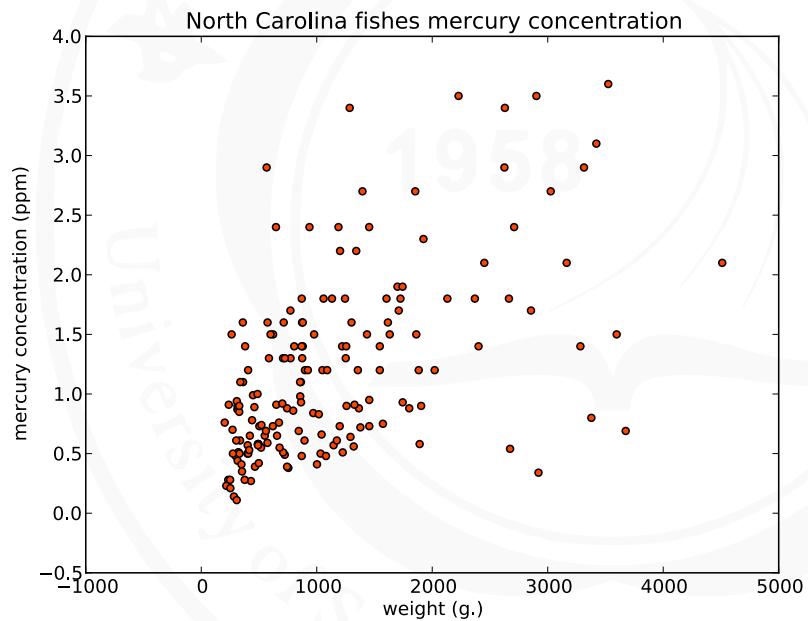
# Plot the points
plt.scatter(data[:,3], data[:,4], c='FF4500')
plt.show()
```

---



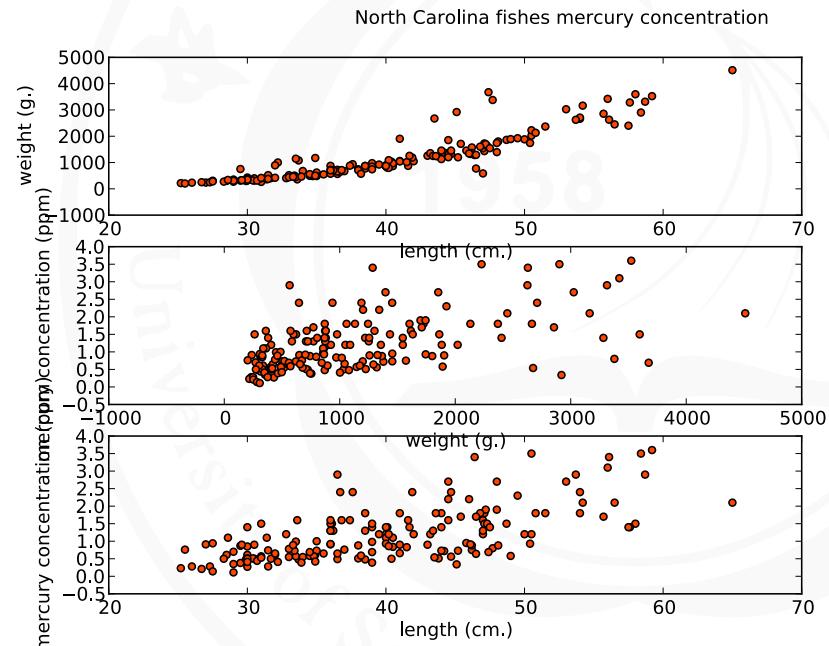
# Mercury & fishes

A first try



# Mercury & fishes

Show more information with *subplots*



# Mercury & fishes

Data from one river with its own color

