# python_lecture_07_plotting_data

March 4, 2020

## 1 Programming with Python

## 2 7 Plotting data

### 2.1 7.1 Line plots

There is no built-in functionality in python to create plots, but thanks to the huge python community, some very sophisticated third-party module were developed to create plots. The module **matplotlib** is amoung other the most popular and most revised plotting module. In order to use it, it needs to be **imported** to your python code. This will look like this.

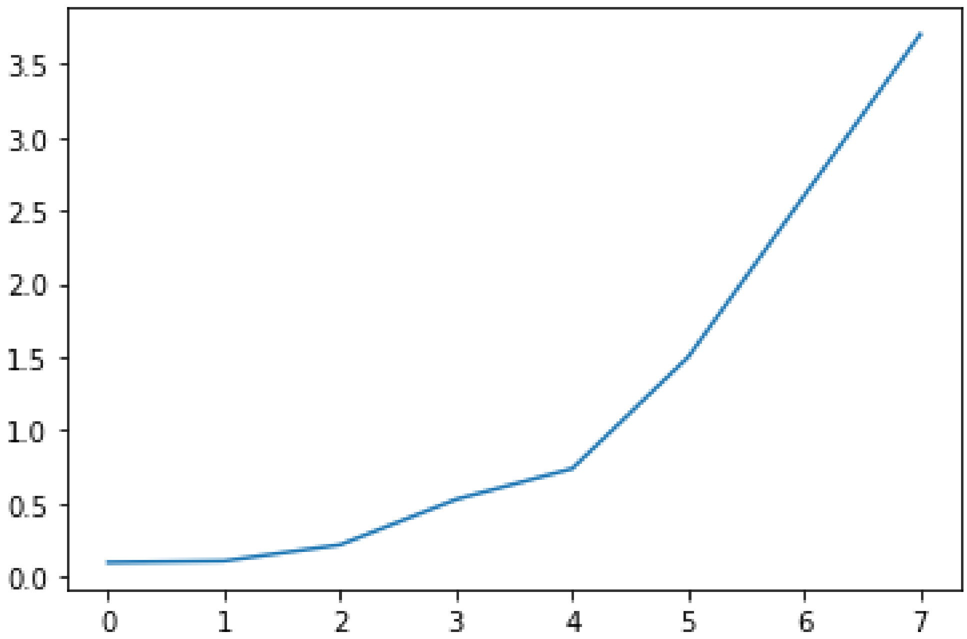
[1]:

```
import matplotlib.pyplot as plt  # package needed for plotting
```

#### plot a single line

[2]:

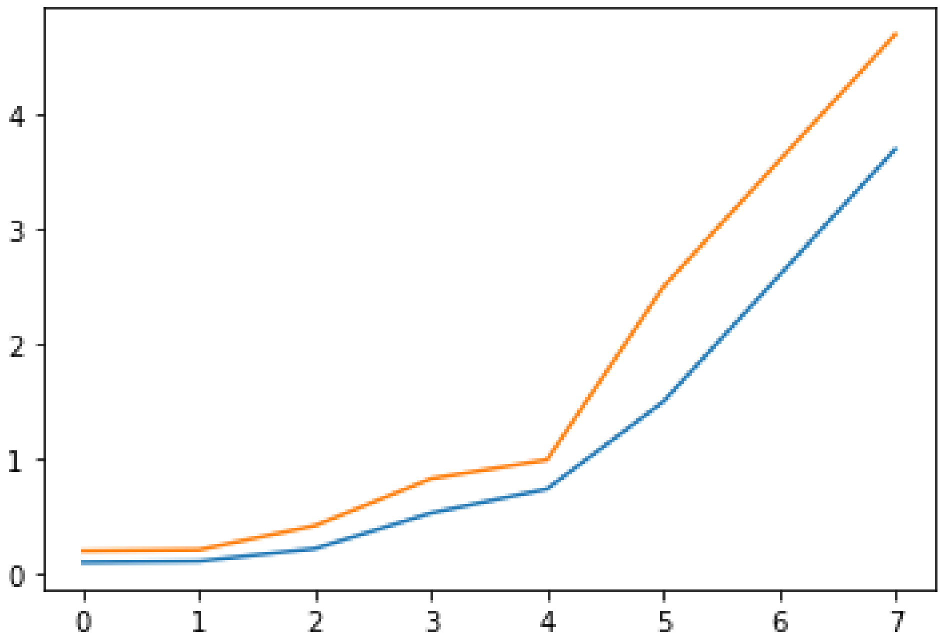
```
# plot a single line

# some random data
x =  [0, 1, 2, 3, 4, 5, 6, 7]
y =  [0.1, 0.11, 0.22, 0.53, 0.74, 1.5, 2.6, 3.7] # the length of y is equal to␣
 ↪x

# plot figure
plt.figure()  # creates empty figure
plt.plot(x, y)  # plot figure

# display figure
plt.show()
```



**plot multiple lines**

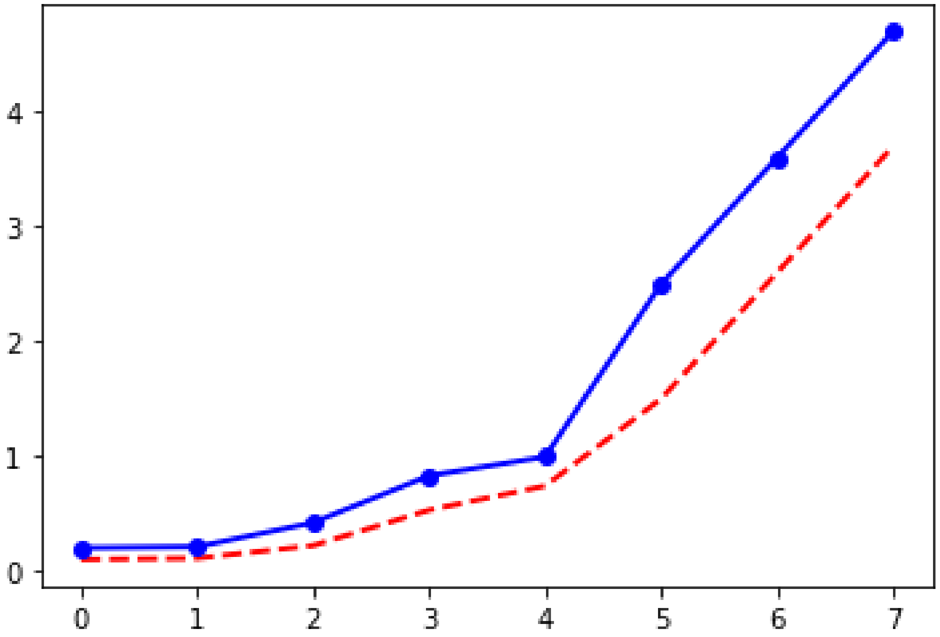
[3]:

```
# plot multiple lines

# some random data
x =  [0, 1, 2, 3, 4, 5, 6, 7]
y =  [0.1, 0.11, 0.22, 0.53, 0.74, 1.5, 2.6, 3.7] # the length of y is equal to x
y2 = [0.2, 0.21, 0.42, 0.83, 0.994, 2.5, 3.6, 4.7]

# plot figure
plt.figure()  # creates empty figure
plt.plot(x, y)  # plot line1 from y
plt.plot(x, y2)  # plot line2 from y

# display figure
plt.show()
```



[4]:

```
# adjust coler and linestyle

# some random data
x =  [0, 1, 2, 3, 4, 5, 6, 7]
y =  [0.1, 0.11, 0.22, 0.53, 0.74, 1.5, 2.6, 3.7] # the length of y is equal to␣
 ↪x
y2 = [0.2, 0.21, 0.42, 0.83, 0.994, 2.5, 3.6, 4.7]

# plot figure
plt.figure()  # creates empty figure
plt.plot(x, y, color="red", linestyle="--", linewidth=2.0)  # plot line1 from y
plt.plot(x, y2, color="blue", marker="o", linewidth=2.0)  # plot line2 from y

# display figure
plt.show()
```



some common marker and line styles:

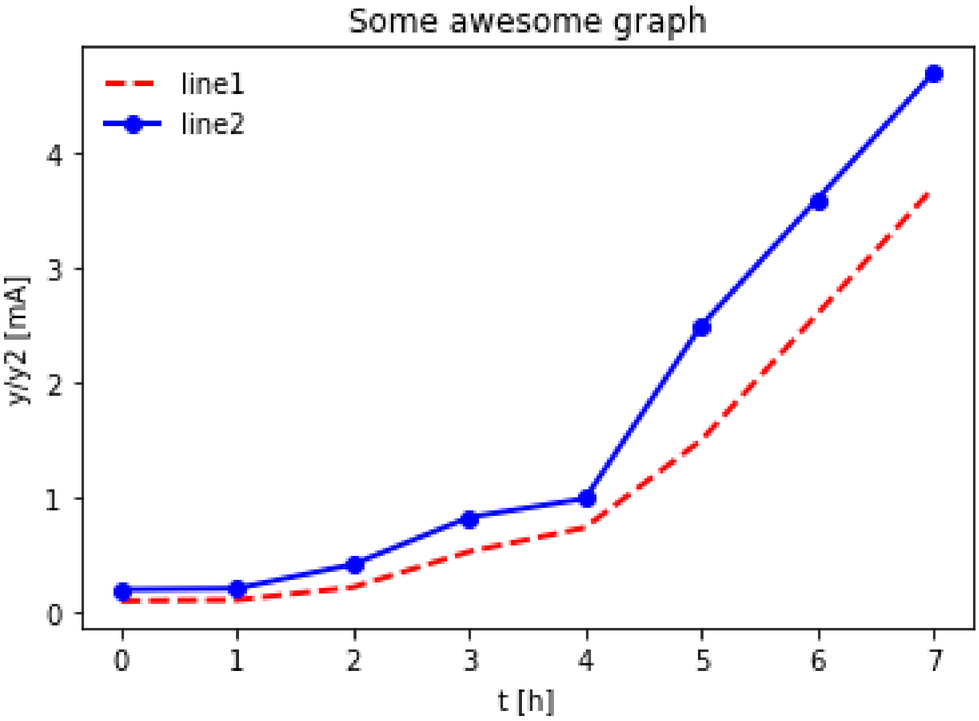
| marker | description |
|---|---|
| . | point |
| o | circle |
| v | triangle down |
| ˆ | trianlge up |
| < | trianlge left |
| > | trianlge right |
| s | square |
| d | small diamond |
| D | big diamond |
| x | x |
|  | star |

| linestyle | description |
|---|---|
| : | dotted |
| - | solid line |
| -. | dash dotted |
| – | dashed |

[5]:

```
# add some labels
```

```python
# some random data
x =  [0, 1, 2, 3, 4, 5, 6, 7]
y =  [0.1, 0.11, 0.22, 0.53, 0.74, 1.5, 2.6, 3.7] # the length of y is equal to x
y2 = [0.2, 0.21, 0.42, 0.83, 0.994, 2.5, 3.6, 4.7]

# plot figure
plt.figure()  # creates empty figure
plt.plot(x, y, color="red", linestyle="--", linewidth=2.0, label = 'line1')  # plot line1 from y
plt.plot(x, y2, color="blue", marker="o", linewidth=2.0, label = 'line2')  # plot line2 from y

# label
plt.legend(loc='upper left', frameon=False) # fremeon adds a frame around the legend if True
plt.ylabel('y/y2 [mA]')
plt.xlabel('t [h]')
plt.title('Some awesome graph')

# display figure
plt.show()
```



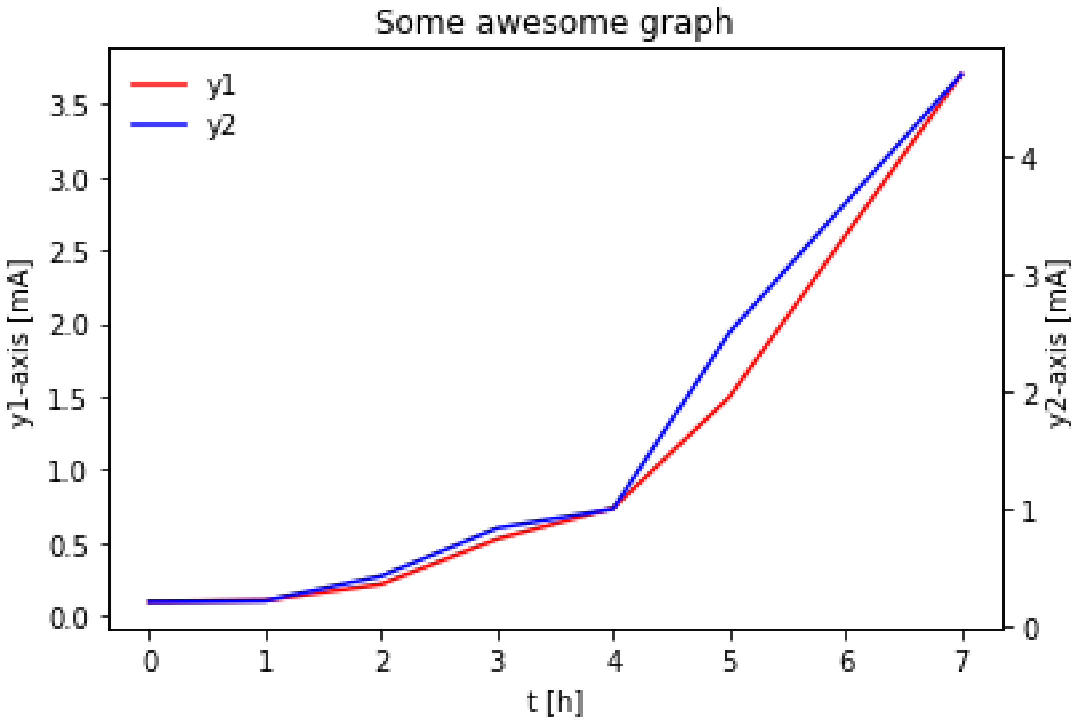
Legend location key can be set to: * best * upper right * upper left * lower left * lower right * right * center left * center right * lower center * upper center * center

Fore more information, check: https://matplotlib.org/users/legend_guide.html

**Use multiple y-axes**

[6]:

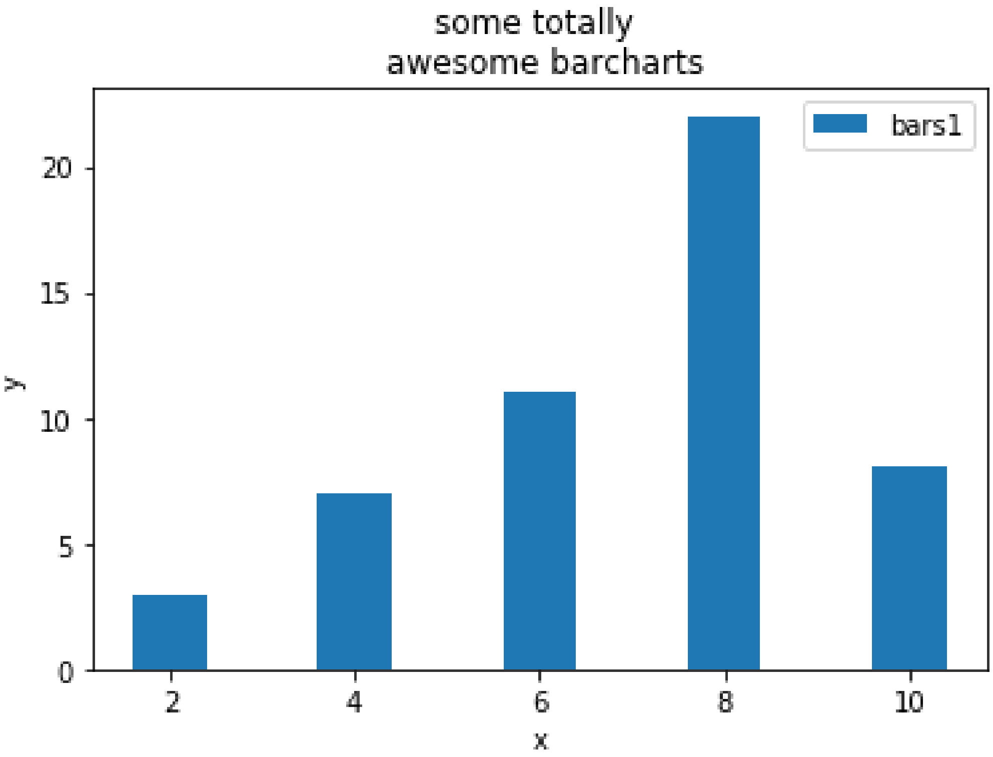
```
# use multiple y-axes

# some random data
x =  [0, 1, 2, 3, 4, 5, 6, 7]
y =  [0.1, 0.11, 0.22, 0.53, 0.74, 1.5, 2.6, 3.7] # the length of y is equal to␣
 ↪x
y2 = [0.2, 0.21, 0.42, 0.83, 0.994, 2.5, 3.6, 4.7]

# plot figure
fig, ax1 = plt.subplots() # create an empty plot with the name fig and an␣
 ↪y-axis ax1
ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis as␣
 ↪ax1
lns1 = ax1.plot(x, y, color="red", label="y1")
lns2 = ax2.plot(x, y2, color="blue", label="y2")

# get labels of plots
lns = lns1+lns2
labs = [l.get_label() for l in lns]

# labels
ax1.legend(lns, labs, loc='upper left', frameon=False)
# ax1.legend(lns, ['y1','y2'], loc='upper left', frameon=False) # you can also␣
 ↪use a manual list
ax1.set_xlabel('t [h]')
ax1.set_ylabel('y1-axis [mA]') # label of first y-axis
ax2.set_ylabel('y2-axis [mA]') # label of second y-axis
plt.title('Some awesome graph')

# display figure
plt.show()
```



## 2.2 7.2 Bar charts

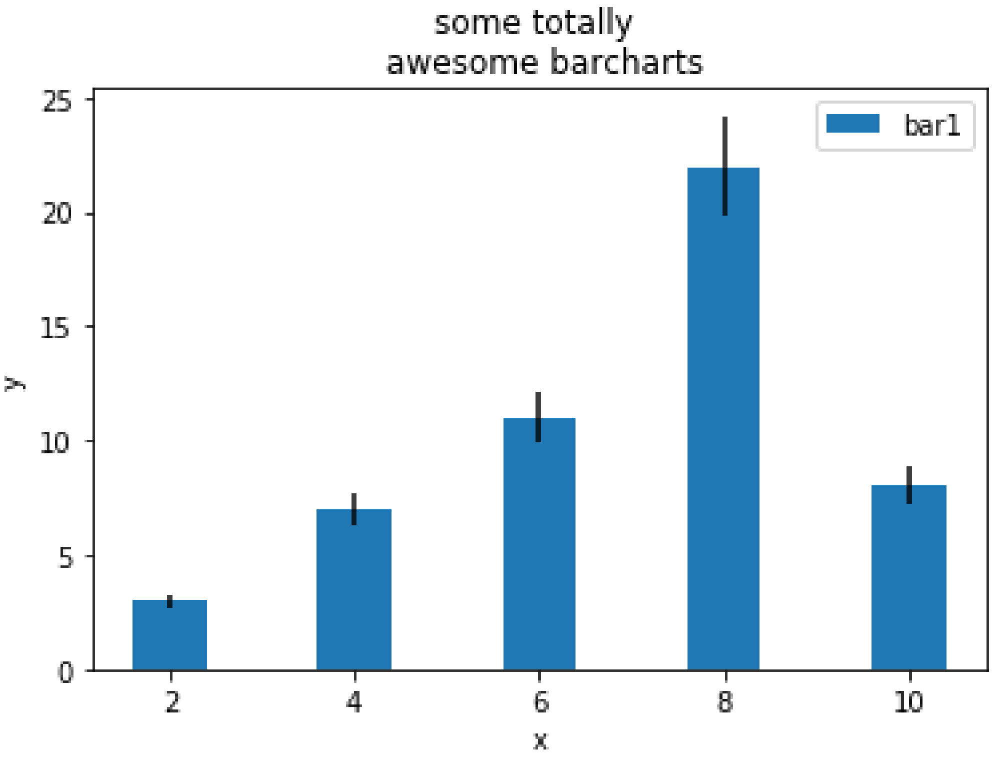
[7]:

```
# some random data
x = [2,4,6,8,10] # represent the x-position where the bar is plotted
y = [3,7,11,22,8] # some values

# plot figure
plt.figure()
plt.bar(x,y, label = 'bars1') # plots the chart

# labels
plt.legend() # shows a legend
plt.xlabel('x') # x label
plt.ylabel('y') # y label
plt.title('some totally \n awesome barcharts') # add a title to the chart

# show figure
plt.show()
```



[8]:

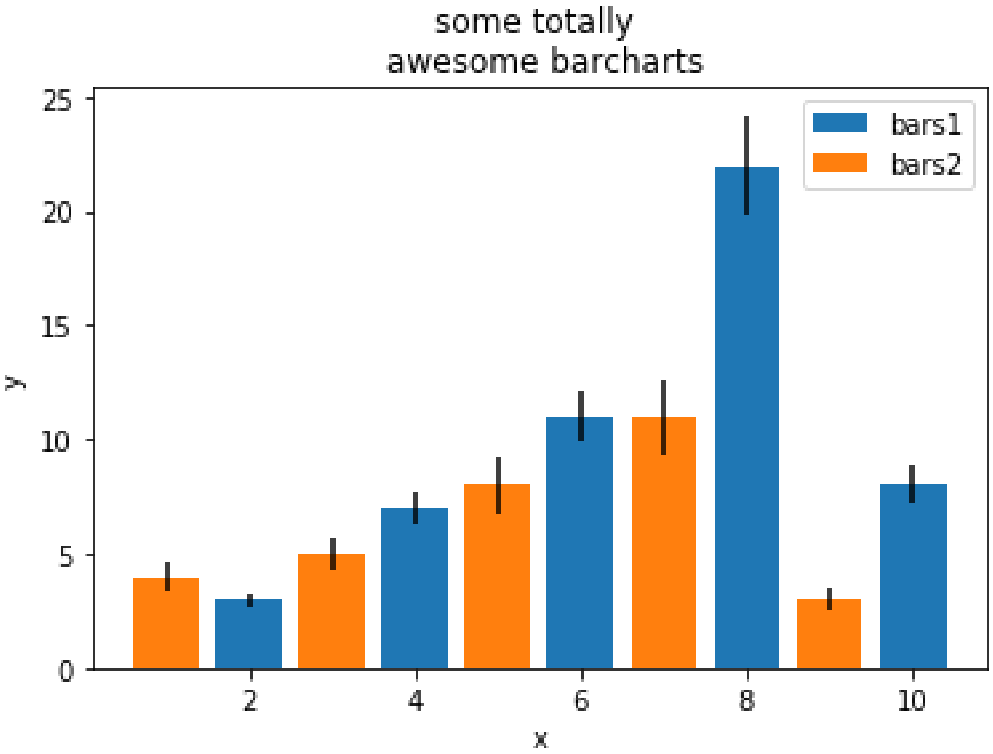
```
# plot with error bar

# some random data
x = [2,4,6,8,10] # represent the x-position where the bar is plotted
y = [3,7,11,22,8] # some values
yerr = [num * 0.1 for num in y]

# plot figure
plt.figure()
plt.bar(x,y, label = 'bar1', yerr = yerr) # plots the chart with error bars

# labels
plt.legend() # shows a legend
plt.xlabel('x') # x label
plt.ylabel('y') # y label
plt.title('some totally \n awesome barcharts') # add a title to the chart

# show figure
plt.show()
```



[9]:

```
# plot with error bar

# some random data
x1 = [2,4,6,8,10] # represent the x-position where the bar is plotted
y1 = [3,7,11,22,8] # some values
y1err = [num * 0.1 for num in y1] # creates a 10% error bar

x2 = [1,3,5,7,9]
y2 = [4,5,8,11,3]
y2err = [num * 0.15 for num in y2] # creates a 15% error bar

# plot figure
plt.figure()
plt.bar(x1,y1, label = 'bars1', yerr = yerr) # plot bars1 into a chart
plt.bar(x2,y2, label = 'bars2', yerr = y2err) # plot bars2 into a chart

# labels
plt.legend() # shows a legend
plt.xlabel('x') # x label
plt.ylabel('y') # y label
plt.title('some totally \n awesome barcharts') # add a title to the chart
```

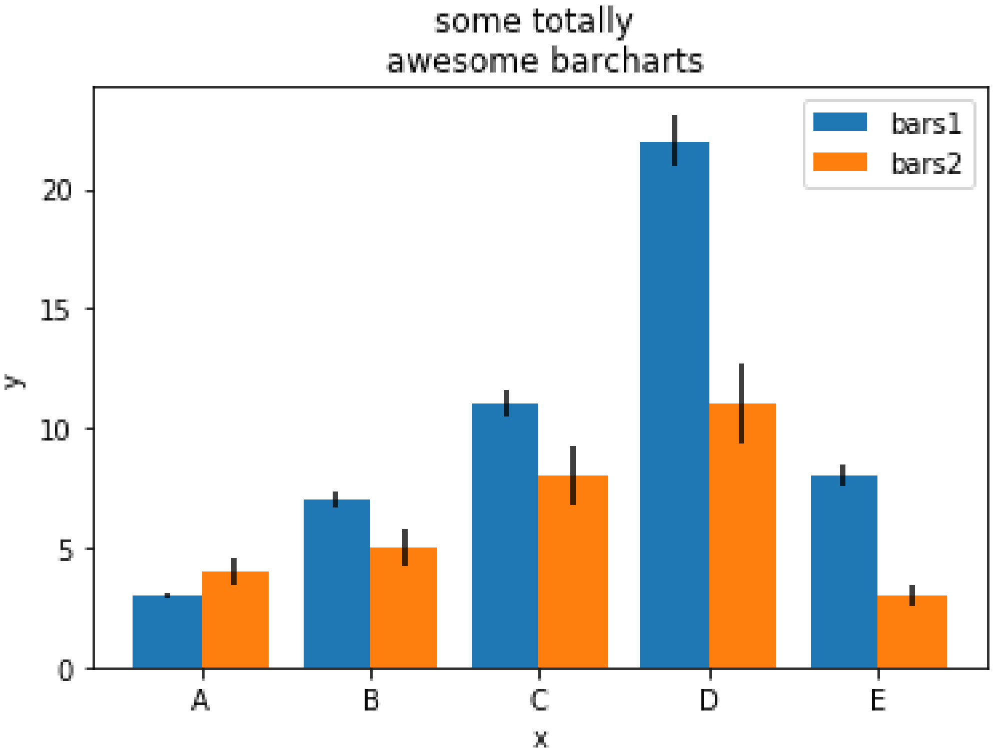
```
# show figure
plt.show()
```



[10]:

```
# plot grouped bars

# some prerequisites for bar charts
bar_width = 0.4 # defines the total area used for that bar in percent

# some random values
x1 = [0,1,2,3,4]
y1 = [3,7,11,22,8]
y1err = [num * 0.05 for num in y1] # creates a 5% error bar

x2 = [num + bar_width for num in x1]
y2 = [4,5,8,11,3]
y2err = [num * 0.15 for num in y2] # creates a 15% error bar

# plot data
plt.figure()
plt.bar(x1,y1, width = bar_width, label = 'bars1', yerr = y1err)
plt.bar(x2,y2, width = bar_width, label = 'bars2', yerr = y2err)
```

```
# labels
xtic = [num + bar_width/2 for num in x1]
plt.xticks(xtic, ['A','B','C','D','E']) # corrects the x tick shift to the right
plt.xlabel('x') # x label
plt.ylabel('y') # y label
plt.title('some totally \n awesome barcharts') # add a title to the chart

# plot legend
plt.legend()

# show plot
plt.show()
```



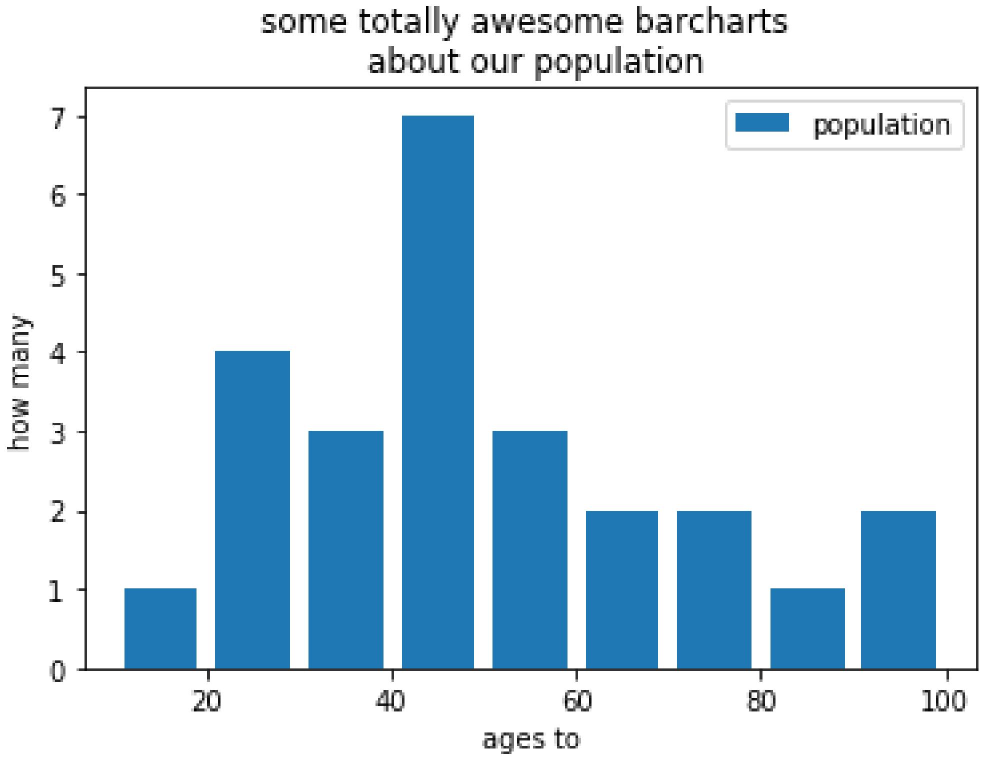
## 3 7.3 Histograms

Histograms consist of bars, but their application is very different. Histograms are usually used to show distributions and bar charts compare categorical values.

[11]:

```
# some random data
population_ages = [2,45,345,76,58,76,67,8,47,56,43,43,42,34,21,14,
                    56,5,67,45,5,34,23,45,23,4,34,23,4,5,7,88,99,95]
```

```
# create bins in order to pack the ages into categories
bins = [10,20,30,40,50,60,70,80,90,100]

# plot figure
plt.figure()
plt.hist(population_ages, bins, rwidth = 0.8, label = 'population') # rwidth␣
 ↪leaves a 20% gap of the total bar width

# labels
plt.xlabel('ages to') # x label
plt.ylabel('how many') # y label
plt.title('some totally awesome barcharts \n about our population') # add a␣
 ↪title to the chart
plt.legend() # shows a legend

# show plot
plt.show()
```

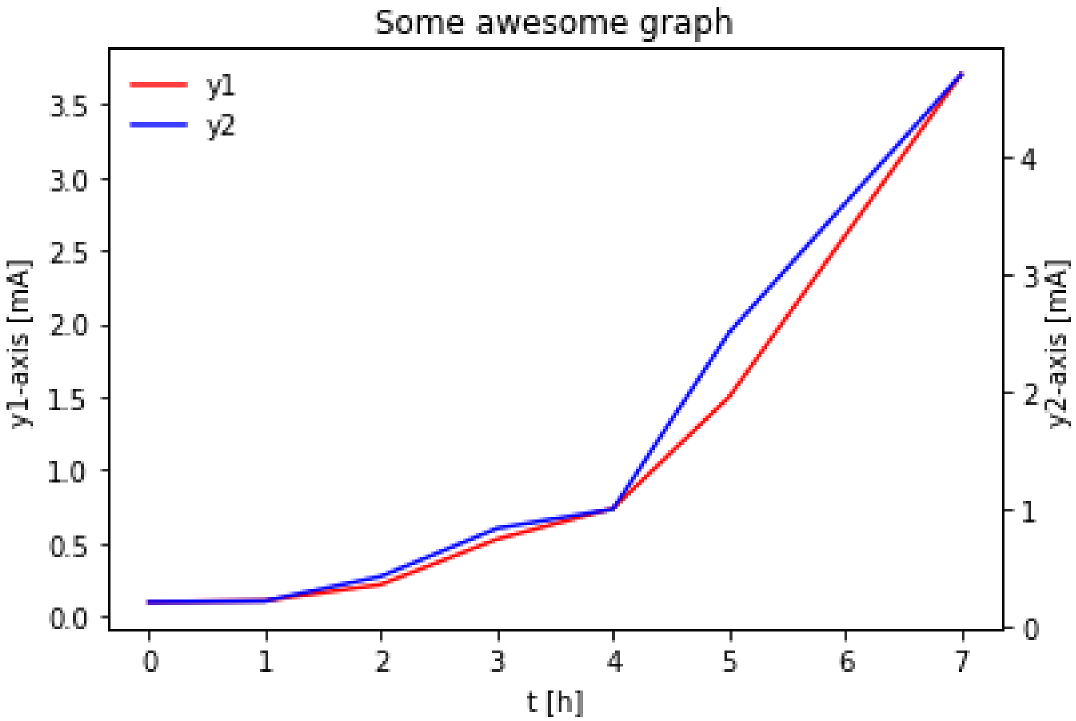


## 3.1 7.4 Saving a figure to disc

Use *plt.savefig('filename')* to save a figure to disc. This can only be done **before plt.show()!**. The file name has to be provided as a string, its extension determines the format, in which the figure

is saved. Common formats include: jpg, png, eps and svg. The availability of individual formats depends on system configuration, and more specifically on the graphics backend used.
For further readings, refer to: https://matplotlib.org/api/_as_gen/matplotlib.pyplot.savefig.html

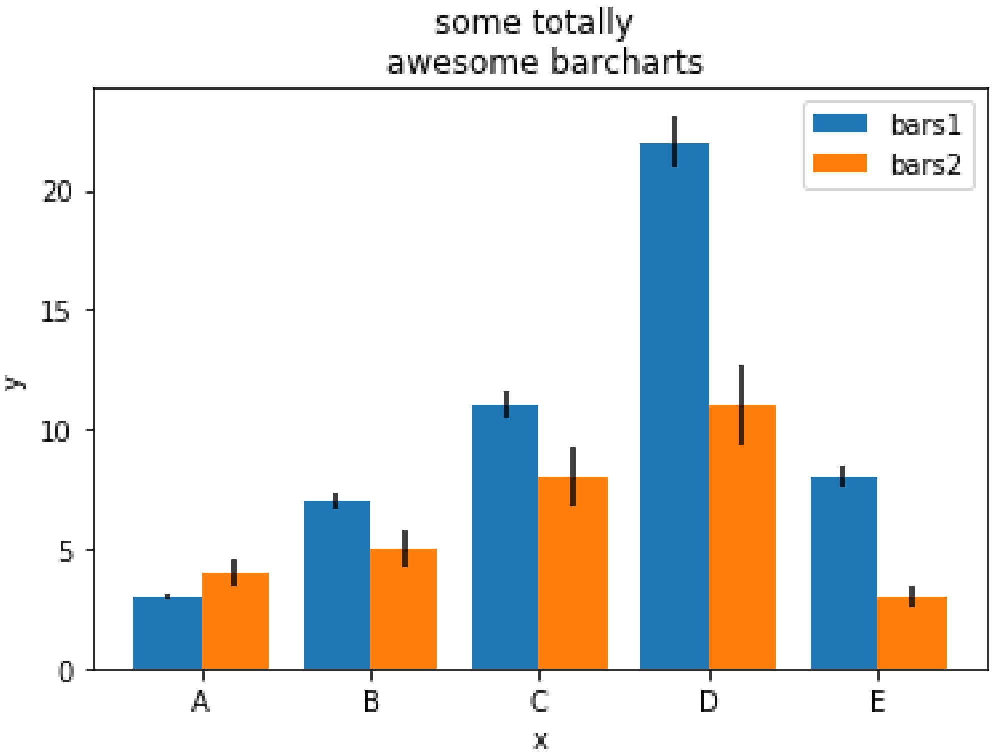
[12]:
```
# save line plot to png

# some random data
x =  [0, 1, 2, 3, 4, 5, 6, 7]
y =  [0.1, 0.11, 0.22, 0.53, 0.74, 1.5, 2.6, 3.7] # the length of y is equal to x
y2 = [0.2, 0.21, 0.42, 0.83, 0.994, 2.5, 3.6, 4.7]

# plot figure
fig, ax1 = plt.subplots() # create an empty plot with the name fig and an y-axis ax1
ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis as ax1
lns1 = ax1.plot(x, y, color="red", label="y1")
lns2 = ax2.plot(x, y2, color="blue", label="y2")

# get labels of plots
lns = lns1+lns2
labs = [l.get_label() for l in lns]

# labels
ax1.legend(lns, labs, loc='upper left', frameon=False)
# ax1.legend(lns, ['y1','y2'], loc='upper left', frameon=False)
ax1.set_xlabel('t [h]')
ax1.set_ylabel('y1-axis [mA]') # label of first y-axis
ax2.set_ylabel('y2-axis [mA]') # label of second y-axis
plt.title('Some awesome graph')

# save to disc
plt.savefig('line2y.png')

# display figure
plt.show()
```



[13]:

```
# save as eps

# some prerequisites for bar charts
bar_width = 0.4 # defines the total area used for that bar in percent

# some random values
x1 = [0,1,2,3,4]
y1 = [3,7,11,22,8]
y1err = [num * 0.05 for num in y1] # creates a 5% error bar

x2 = [num + bar_width for num in x1]
y2 = [4,5,8,11,3]
y2err = [num * 0.15 for num in y2] # creates a 15% error bar

# plot data
plt.figure()
plt.bar(x1,y1, width = bar_width, label = 'bars1', yerr = y1err)
plt.bar(x2,y2, width = bar_width, label = 'bars2', yerr = y2err)

# labels
xtic = [num + bar_width/2 for num in x1]
plt.xticks(xtic, ['A','B','C','D','E']) # corrects the x tick shift to the right
plt.xlabel('x') # x label
plt.ylabel('y') # y label
plt.title('some totally \n awesome barcharts') # add a title to the chart

# plot legend
plt.legend()

# save plot as eps
plt.savefig('barchart.eps', format = 'eps')

# show plot
plt.show()
```

```
The PostScript backend does not support transparency; partially transparent
artists will be rendered opaque.
The PostScript backend does not support transparency; partially transparent
artists will be rendered opaque.
```



### 3.1.1 Further reading:

for further reference check: https://matplotlib.org/tutorials/index.html