CMM201 Week 11: Plotting Images in Python

1 Plotting Data in Python

1.1 Aims of the Lecture

- Learn how to do basic plots using *numpy arrays* and the *matplotlib* package

1.2 Additional Reading

- Matplotlib documentation
- Matplotlib tutorial
- Matplotlib pie chart tutorial

1.3 Using Artificial Data

- First, we will install the *matplotlib* package. Maybe you already have it from the "images" lecture (week 9)

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor2

```
!pip install matplotlib
```

```
Requirement already satisfied: matplotlib in c:\anaconda\lib\site-packages
(3.0.3)
Requirement already satisfied: numpy>=1.10.0 in c:\anaconda\lib\site-packages
(from matplotlib) (1.16.2)
Requirement already satisfied: cycler>=0.10 in c:\anaconda\lib\site-packages
(from matplotlib) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\anaconda\lib\site-
packages (from matplotlib) (1.0.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
c:\anaconda\lib\site-packages (from matplotlib) (2.3.1)
Requirement already satisfied: python-dateutil>=2.1 in c:\anaconda\lib\site-
packages (from matplotlib) (2.8.0)
Requirement already satisfied: six in c:\anaconda\lib\site-packages (from
cycler>=0.10->matplotlib) (1.12.0)
Requirement already satisfied: setuptools in c:\anaconda\lib\site-packages (from
kiwisolver>=1.0.1->matplotlib) (40.8.0)
```

- The function that we will use from *matplotlib* is called **pyplot**.

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor2

```
import matplotlib.pyplot as plt
```

1.3.1 Line Plots

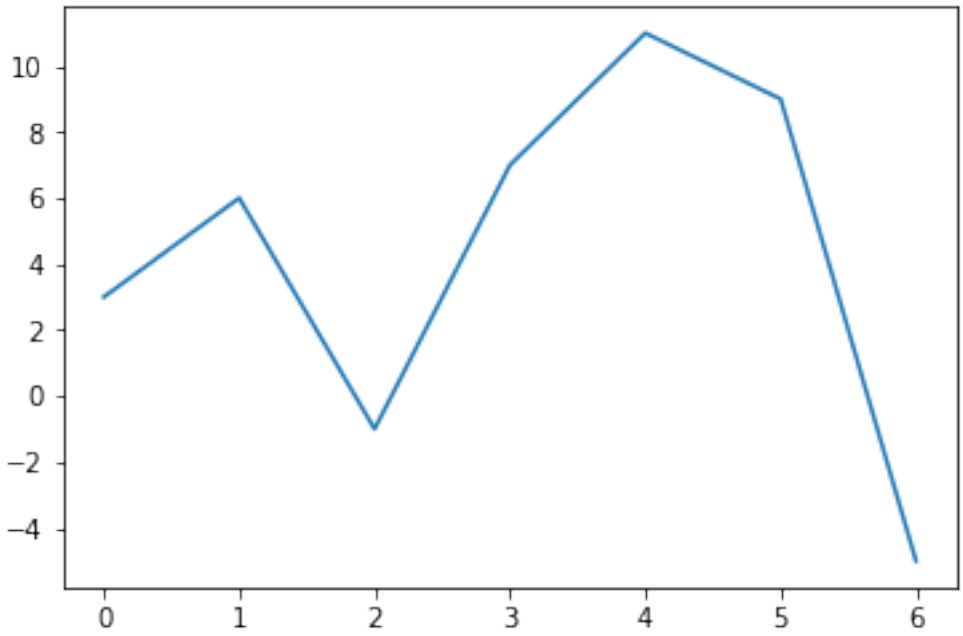
- You can create a line plot by defining a list
- By default, the x-axis will have integer values starting in 0.

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor6

```
plt.plot([3,6,-1,7,11,9,-5])
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0] Outoutcolor6

```
[<matplotlib.lines.Line2D at 0x1d2cc70ce80>]
```



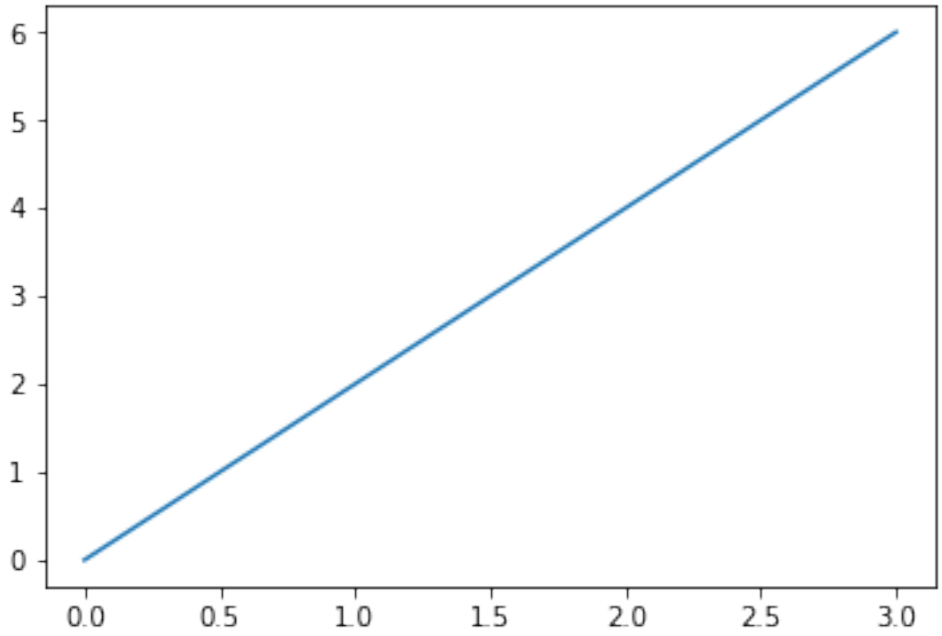
- You can define two lists and then plot them (both have to be the same size).

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor10

```
plt.plot([0,1,2,3],[0,2,4,6])
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0] Outoutcolor10

```
[<matplotlib.lines.Line2D at 0x14ea91eb978>]
```

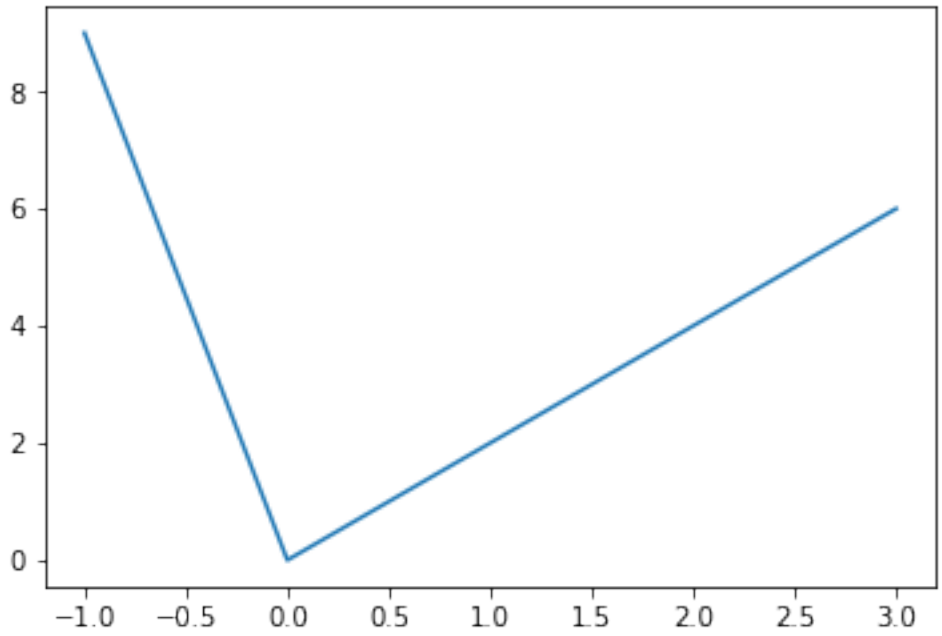


[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor12

```
plt.plot([-1,0,1,2,3],[9,0,2,4,6])
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0] Outoutcolor12

```
[<matplotlib.lines.Line2D at 0x14ea9082048>]
```

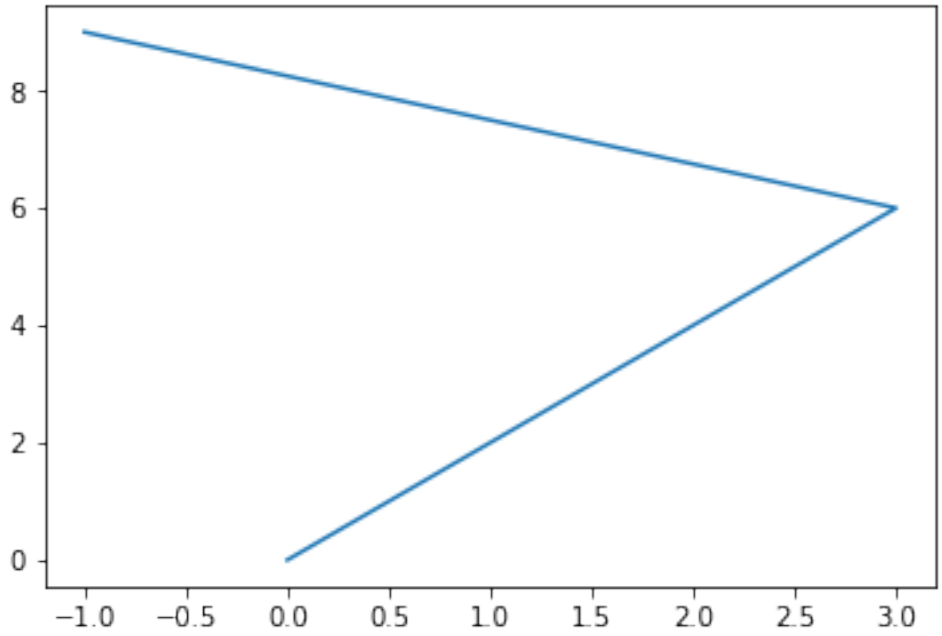


[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor11

```
# Inverting the order draws the line differently
plt.plot([0,1,2,3,-1],[0,2,4,6,9])
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0] Outoutcolor11

```
[<matplotlib.lines.Line2D at 0x14e86b63cc0>]
```

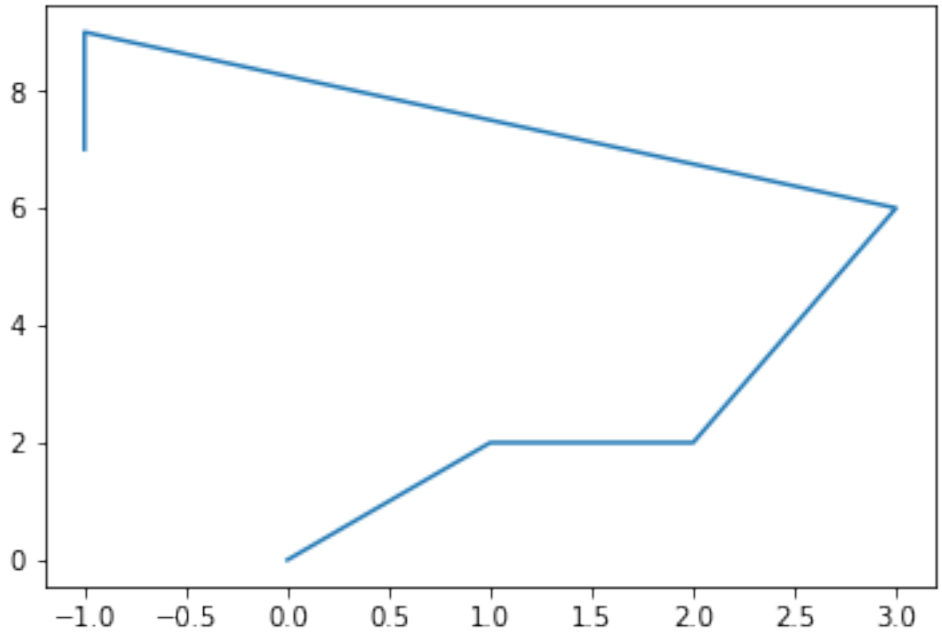


[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor13

```
# You can repeat values for x and y
plt.plot([0,1,2,3,-1,-1],[0,2,2,6,9,7])
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0] Outoutcolor13

```
[<matplotlib.lines.Line2D at 0x14ea9292eb8>]
```



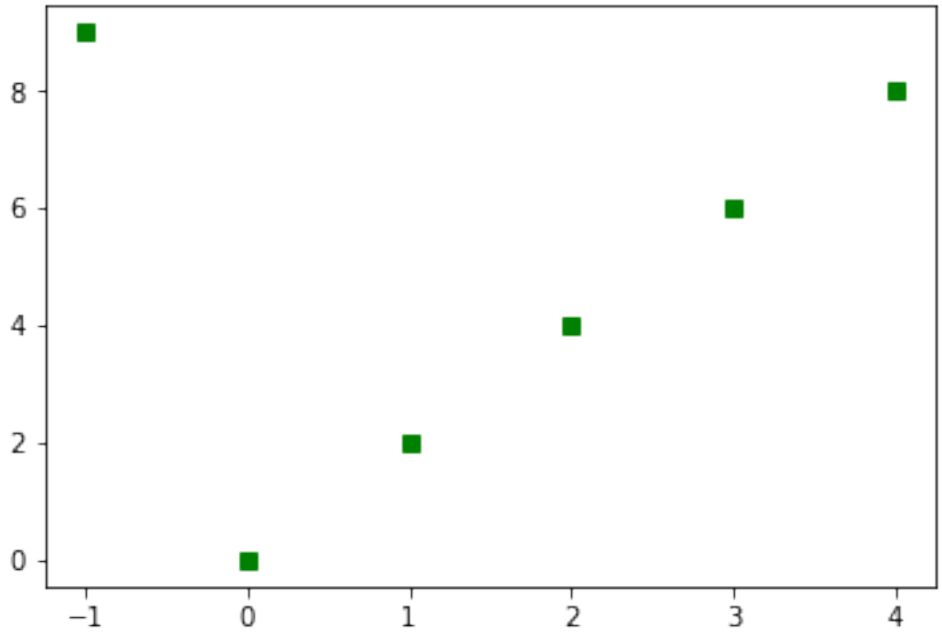
- By default plots are made with a blue solid line, but this can be changed:

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor20

```
# You can use the following:
# blue (b), red (r), green (g)...
# line (-), square (s), dash (--), triangle (^)..
plt.plot([-1,0,1,2,3,4,],[9,0,2,4,6,8],'gs')
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0] Outoutcolor20

```
[<matplotlib.lines.Line2D at 0x1d2ce3ad400>]
```



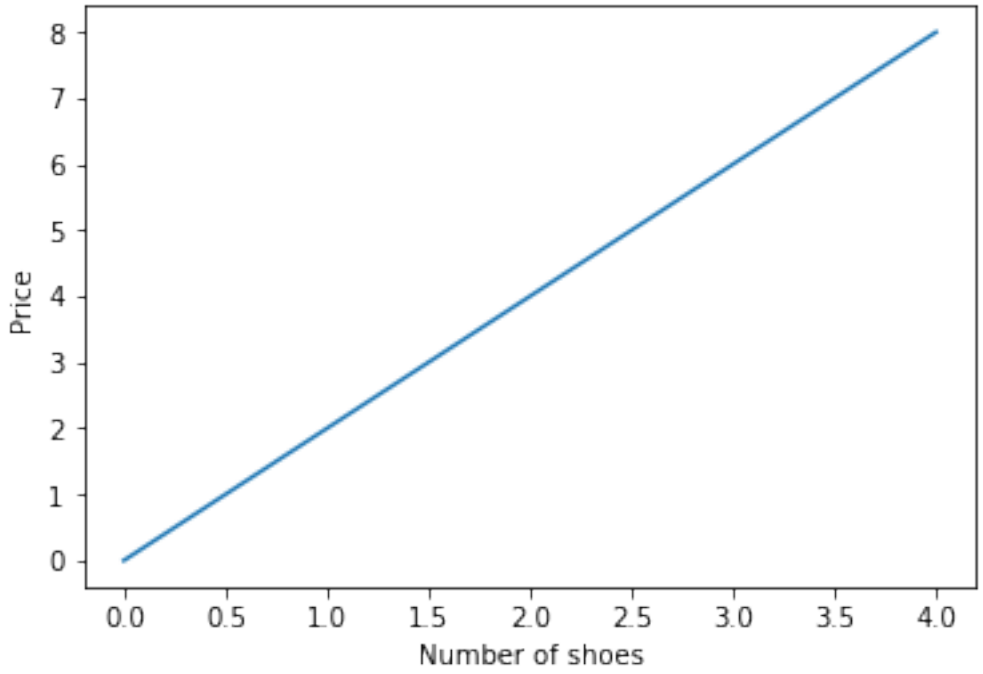
- Adding x- and y-axis labels

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor21

```
plt.plot([0,1,2,3,4],[0,2,4,6,8])
plt.xlabel('Number of shoes')
plt.ylabel('Price')
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0] Outoutcolor21

```
Text(0, 0.5, 'Price')
```



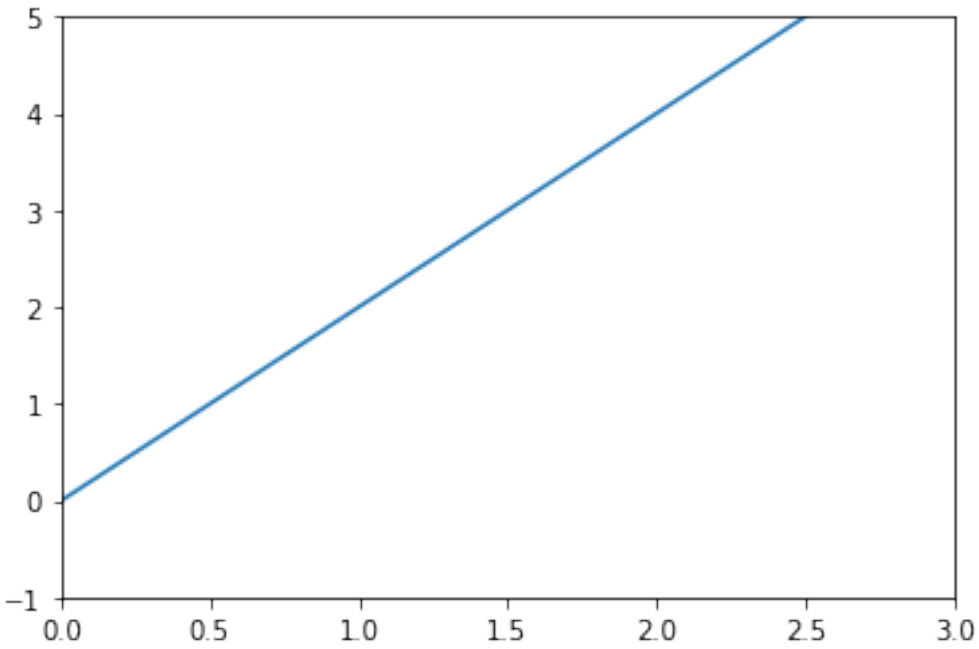
- By using *plt.axis()*, you can set the minimum and maximum values of the *x*- and *y*-axis:

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor9

```
plt.plot([0,1,2,3,4],[0,2,4,6,8])
plt.axis([0,3,-1,5])
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0] Outoutcolor9

```
[0, 3, -1, 5]
```

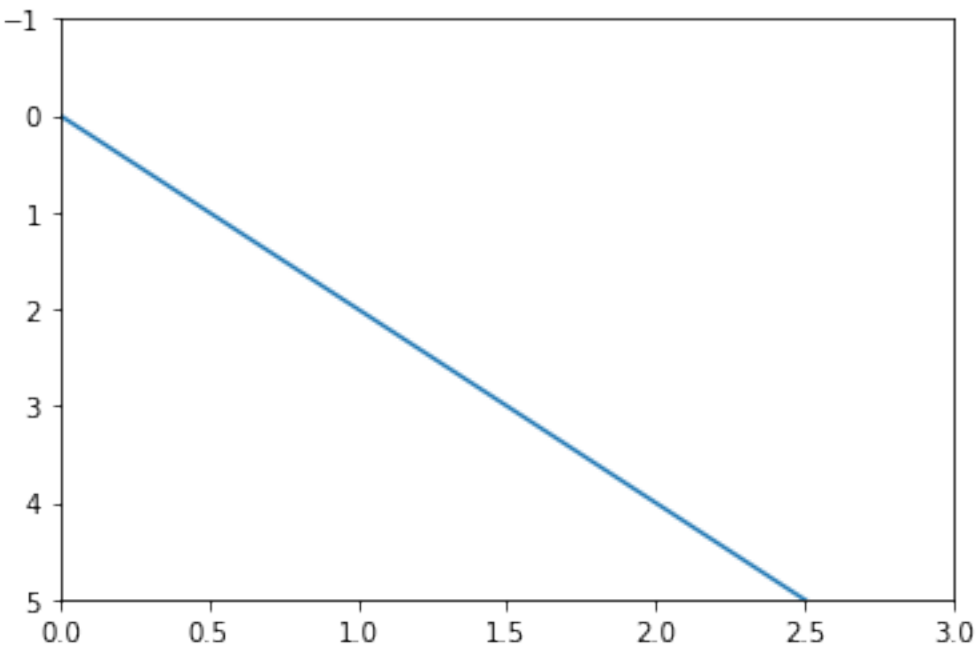


[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor24

```
plt.plot([0,1,2,3,4],[0,2,4,6,8])
# If you pùt the third number larger than the fourth
# Then the graph inverts
plt.axis([0,3,5,-1])
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0] Outoutcolor24

```
[0, 3, 5, -1]
```



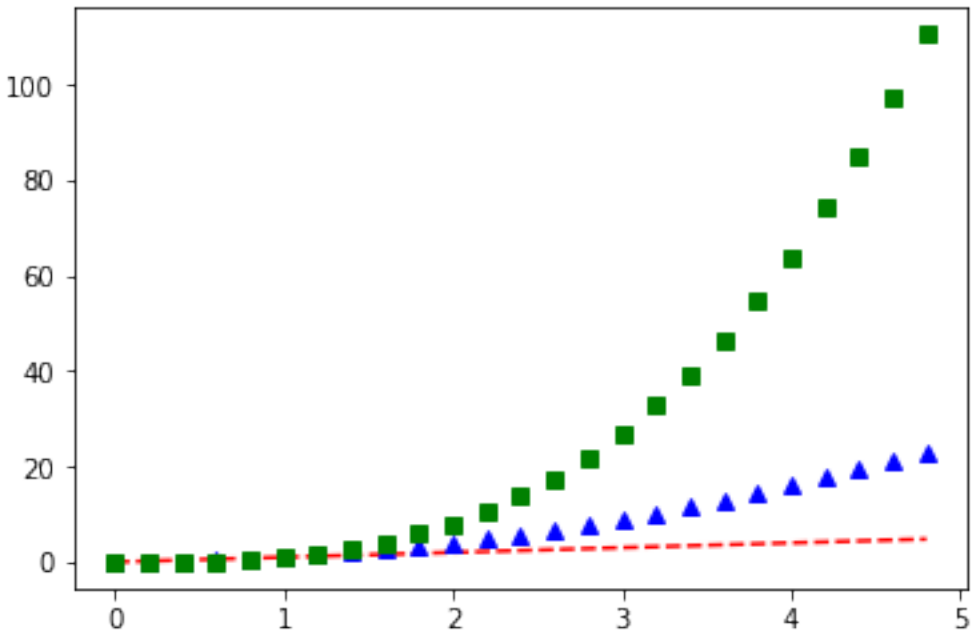
- You can also use *numpy* arrays to plot values.
- This is better than using lists, as you can do calculations with the values.

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor14

```
import numpy as np
t=np.arange(0,5,0.2)
#print(t)
plt.plot(t,t,'r--',t,t**2,'b^',t,t**3,'gs')
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0] Outoutcolor14

```
[<matplotlib.lines.Line2D at 0x14eaa3590f0>,
 <matplotlib.lines.Line2D at 0x14eaa359358>,
 <matplotlib.lines.Line2D at 0x14eaa359ac8>]
```

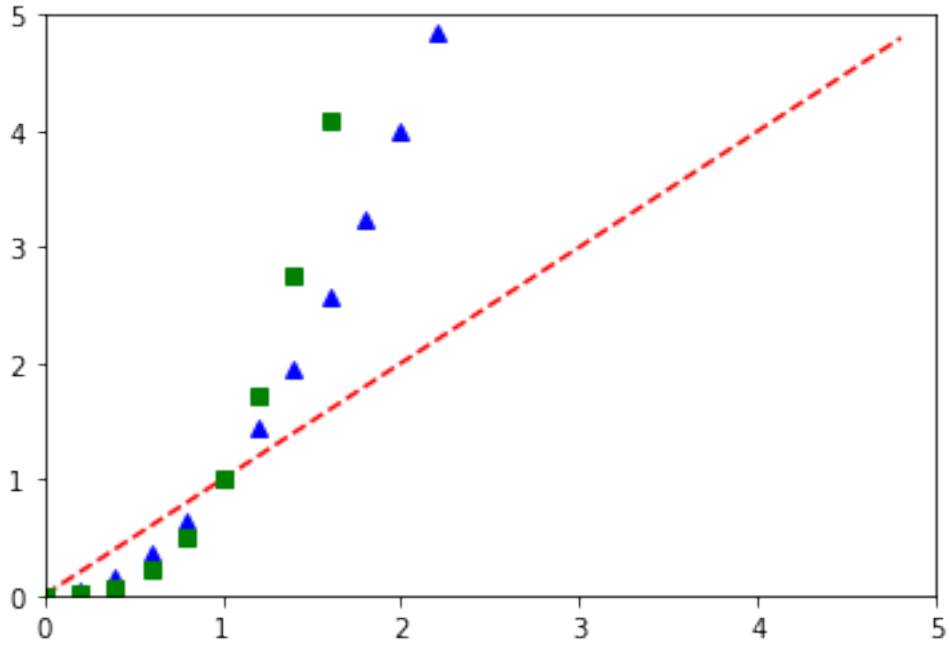


[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor27

```
# Adding axis specification to "zoom in"
import numpy as np
t=np.arange(0,5,0.2)
#print(t)
plt.plot(t,t,'r--',t,t**2,'b^',t,t**3,'gs')
plt.axis([0,5,0,5])
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0] Outoutcolor27

```
[0, 5, 0, 5]
```



1.3.2 Scatter Plots

- Scatter plots are used to plot data along the coordinate plane.
- These are really useful when you want to analyse data trends.
- To test scatterplots, we will create a dictionary with four ranges of numbers by using *numpy* and the *random* function.

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor35

```
data={'a':np.arange(10),
'b':10*np.random.randn(10),
'c':np.random.randint(0,100,10),
'd':np.abs(np.random.randn(10))*100}
print(data)
```

```
{'a': array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), 'b': array([ -0.85027867,
-1.73238981, -19.70246247,   2.91261581,
         8.47249125,   8.31288618,  -4.30196017,  -3.56844447,
        -0.33327018,   4.24613285]), 'c': array([95, 58, 28, 44, 25,  1, 37, 74,
15, 49]), 'd': array([ 85.21847445,   5.5768126 ,  96.20911555, 129.7786845 ,
        59.77628552,  31.2037584 ,  40.19301022,  38.6570193 ,
        36.82334713,  12.43878839])}
```

- Now we can plot the data using

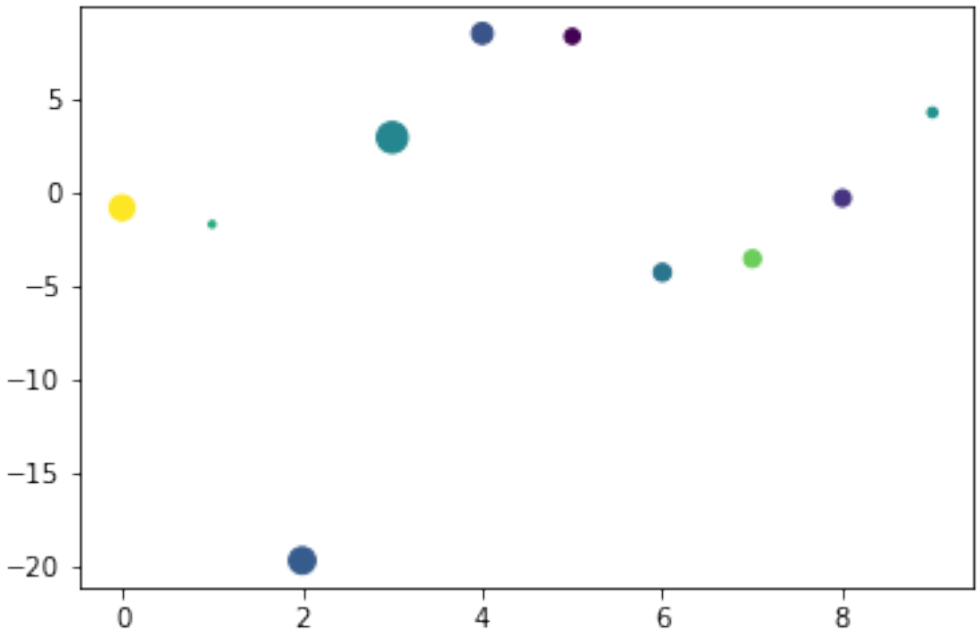
- a as the x-axis values
- b as the y-axis values
- c as different colours for the data (these colours are random according to the number)
- d as different sizes/weights

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor36

```
plt.scatter(data['a'],data['b'],c=data['c'],s=data['d'])
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0] Outoutcolor36

```
<matplotlib.collections.PathCollection at 0x1d2ce4750b8>
```

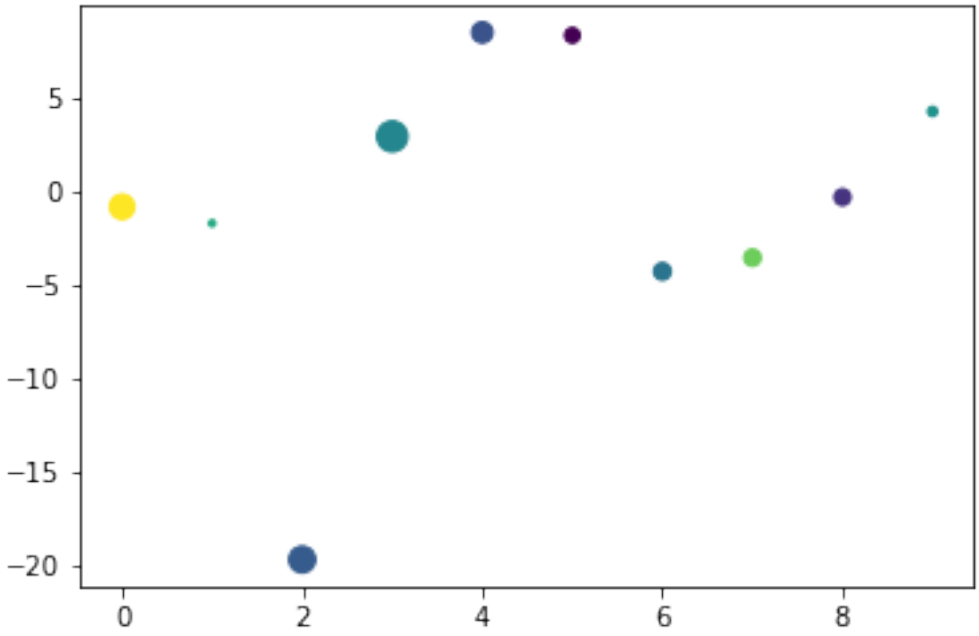


[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor37

```
# Another option is to use only the keys
# and specify the dictionary  as the data to be used
plt.scatter('a','b',c='c',s='d',data=data)
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0] Outoutcolor37

```
<matplotlib.collections.PathCollection at 0x1d2ce58a2b0>
```



Can you think of any "real life" example of data where you can use this?

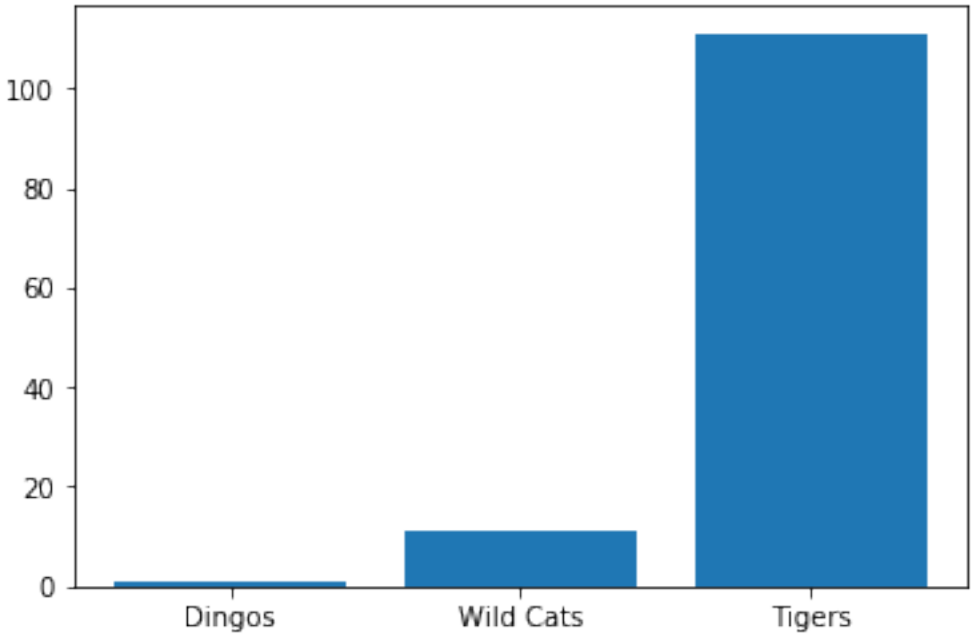
Fig 1. An example of a scatter plot

1.3.3 Categorical Data

- Using two lists, you can produce charts with categorical data

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor43

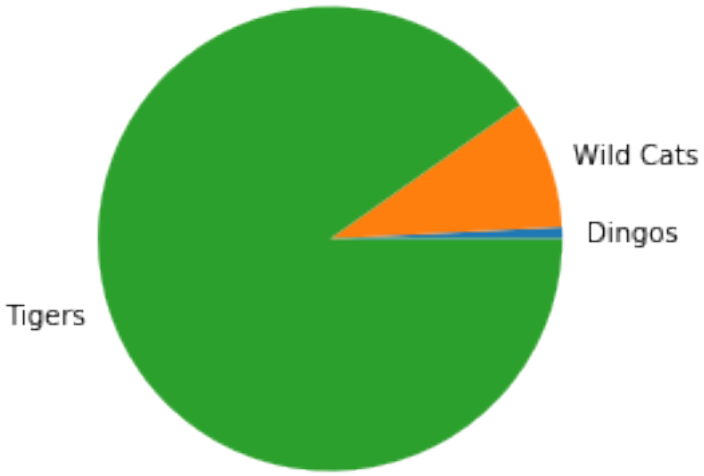
```
names=["Dingos","Wild Cats","Tigers"]
values=[1,11,111]
#plt.figure()
plt.bar(names,values) # Create a bar chart
plt.show() # show the figure
```



- Pie chart

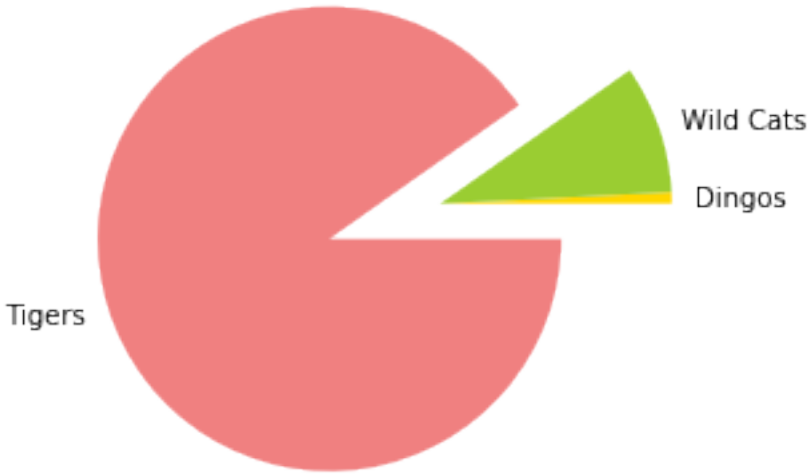
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, col-frame=cellborder] Inincolor45

```
plt.pie(values,labels=names)
plt.show()
```



[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor49

```
c = ['gold', 'yellowgreen', 'lightcoral']
e = (0, 0, 0.5)  # separate third slice
plt.pie(values,labels=names,explode=e,colors=c)
plt.show()
```



1.4 Plotting the IRIS Dataset

- Once again, we will load the IRIS dataset and save all the contents in different variables:
- The IRIS dataset has four value columns (sepal/petal-length/width) and a class/target.

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor1

```
## Load iris dataset
from sklearn import datasets
iris = datasets.load_iris()
data = iris['data']
header = iris['feature_names']
target = iris['target']
target_names = iris['target_names']
```

- Tally the target/class to see how many samples of each plant

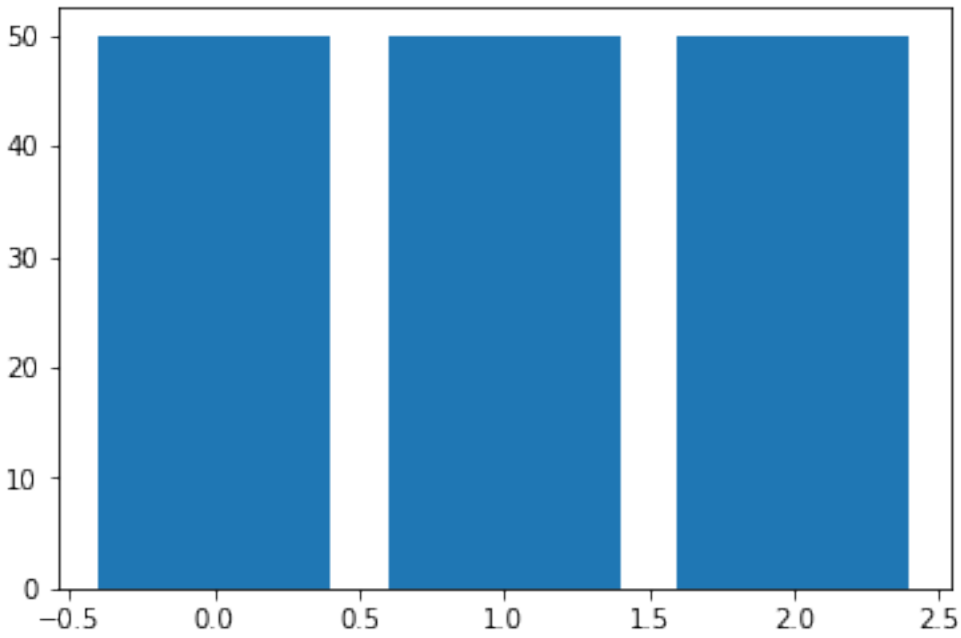
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor51

```
unique_elements, counts_elements = np.unique(target, return_counts=True)
print(unique_elements, counts_elements)
```

```
[0 1 2] [50 50 50]
```

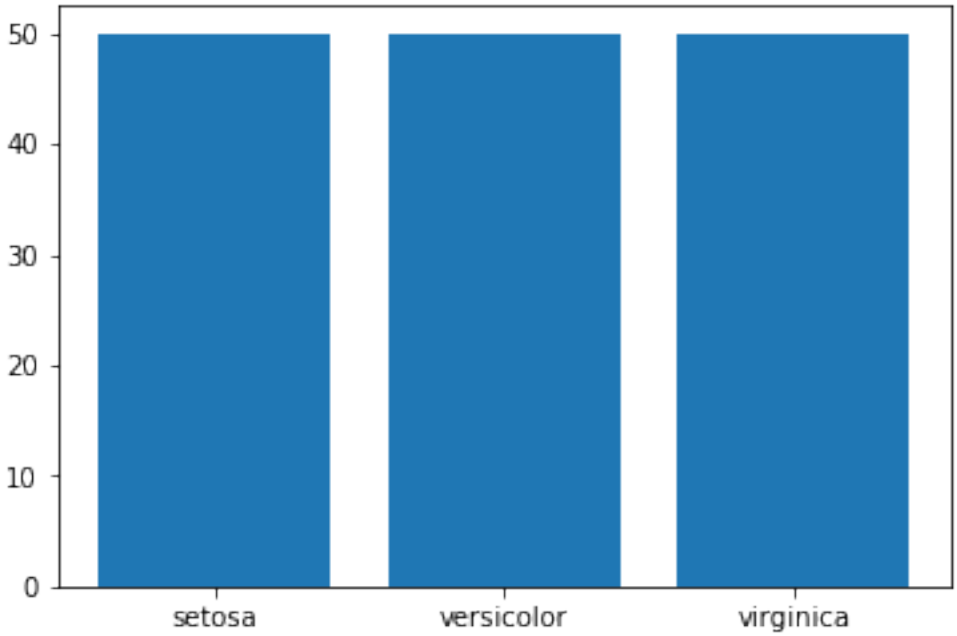
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor53

```
# Plot a bar chart with the tally
plt.bar(unique_elements,counts_elements)
plt.show()
```



[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor54

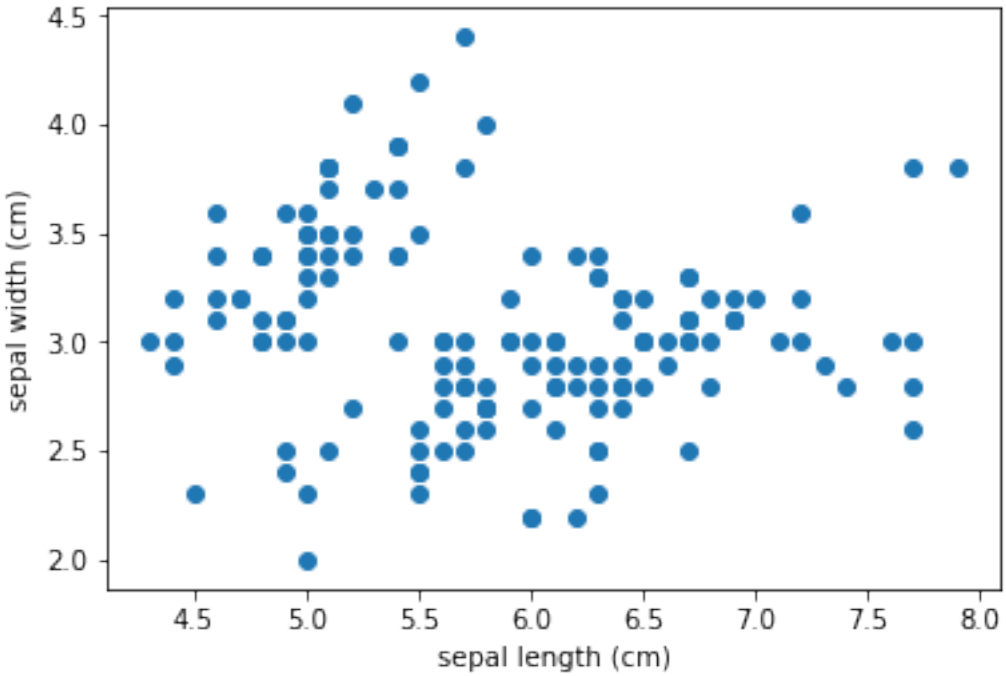
```
# Using the target names instead
plt.bar(target_names,counts_elements)
plt.show()
```



- Plotting variables against each other (scatter)

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor55

```
plt.scatter(data[:,0],data[:,1])
plt.xlabel(header[0])
plt.ylabel(header[1])
plt.show()
```



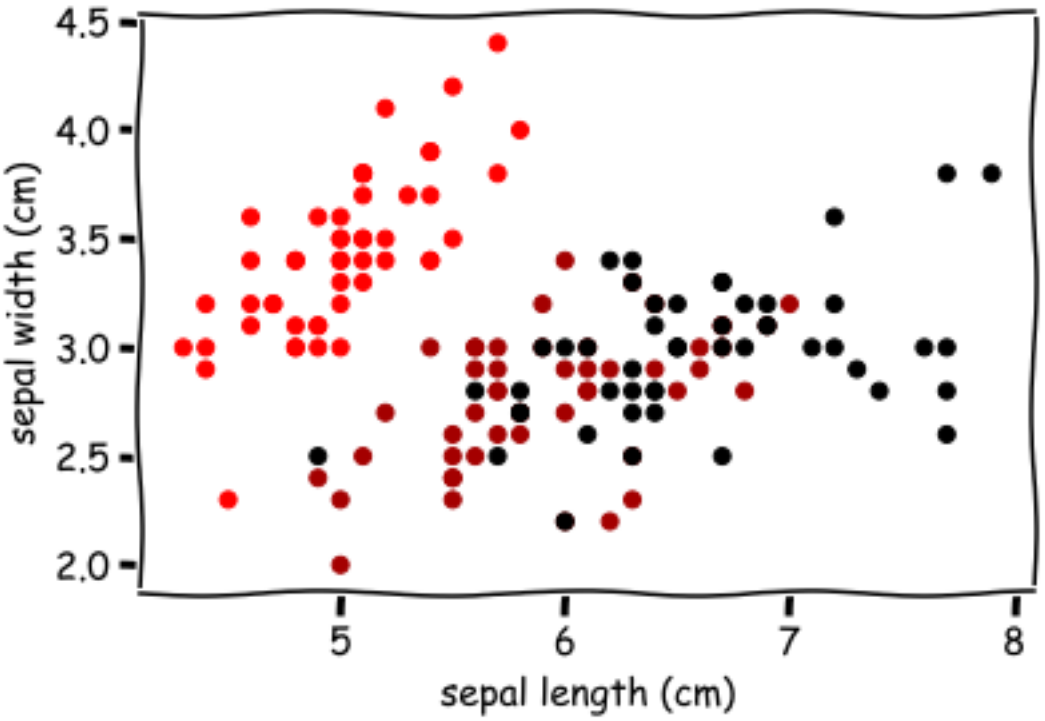
- Using the target as a colour differentiator

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor62

```
plt.scatter(data[:,0],data[:,1],c=target,cmap='flag')
plt.xlabel(header[0])
plt.ylabel(header[1])
plt.xkcd() ## This creates you rplots with the pencil-like format
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0] Outoutcolor62

```
<matplotlib.rc_context at 0x1d2d0d0b278>
```



- Using a third variable as a size differentiator

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor65

```
## This is the data of the third column.
## Notice values are not too different from each other!
print(data[:,2])
```

```
[1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 1.5 1.6 1.4 1.1 1.2 1.5 1.3 1.4
 1.7 1.5 1.7 1.5 1.  1.7 1.9 1.6 1.6 1.5 1.4 1.6 1.6 1.5 1.5 1.4 1.5 1.2
 1.3 1.4 1.3 1.5 1.3 1.3 1.3 1.6 1.9 1.4 1.6 1.4 1.5 1.4 4.7 4.5 4.9 4.
 4.6 4.5 4.7 3.3 4.6 3.9 3.5 4.2 4.  4.7 3.6 4.4 4.5 4.1 4.5 3.9 4.8 4.
 4.9 4.7 4.3 4.4 4.8 5.  4.5 3.5 3.8 3.7 3.9 5.1 4.5 4.5 4.7 4.4 4.1 4.
 4.4 4.6 4.  3.3 4.2 4.2 4.2 4.3 3.  4.1 6.  5.1 5.9 5.6 5.8 6.6 4.5 6.3
 5.8 6.1 5.1 5.3 5.5 5.  5.1 5.3 5.5 6.7 6.9 5.  5.7 4.9 6.7 4.9 5.7 6.
 4.8 4.9 5.6 5.8 6.1 6.4 5.6 5.1 5.6 6.1 5.6 5.5 4.8 5.4 5.6 5.1 5.1 5.9
 5.7 5.2 5.  5.2 5.4 5.1]
```

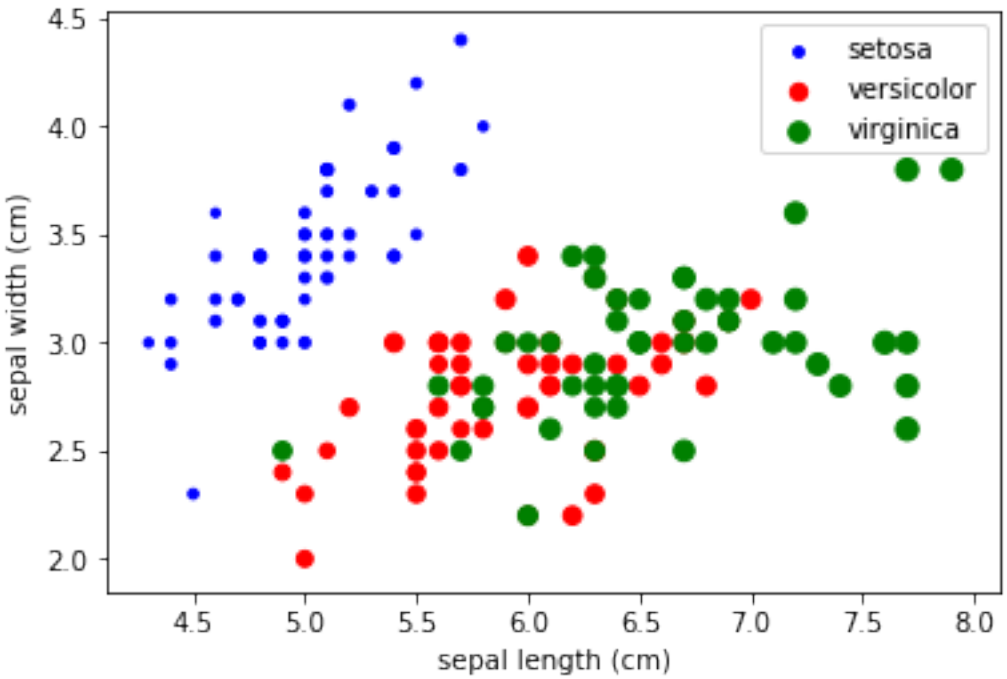
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor8

```
## Plotting a scatterplot with a legend
## Notice that we multiply s*10 to make sizes difference larger
```

```
## You need to create one scatterplot per category
setosa = plt.scatter(data[0:50][:,0],data[0:50][:,1],c='b',s=data[0:50][:,2]*10)
versicolor = plt.scatter(data[50:100][:,0],data[50:100][:,1],c='r',s=data[50:100][:,2]*10)
virginica = plt.scatter(data[100:150][:,0],data[100:150][:,1],c='g',s=data[100:150][:,2]*10)

# Then you create the legend
plt.legend((setosa,versicolor,virginica),target_names,loc='upper right')

# Finally you put labels to the axis and show
plt.xlabel(header[0])
plt.ylabel(header[1])
plt.show()
```



- 3D Plots

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] Inincolor15

```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(1, figsize=(4, 3))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
for name, label in [('Setosa', 0),('Versicolor', 1),('Virginica', 2)]:
    ax.text3D(data[target == label, 3].mean(),
              data[target == label, 0].mean(),
```

```
                data[target == label, 2].mean() + 2, name,
                horizontalalignment='center',
                bbox=dict(alpha=.2, edgecolor='w', facecolor='w'))
ax.scatter(data[:, 3],data[:, 0],data[:, 2],c=target.astype(np.float),edgecolor='k')
ax.w_xaxis.set_ticklabels([])
ax.w_yaxis.set_ticklabels([])
ax.w_zaxis.set_ticklabels([])
ax.set_xlabel('Petal width')
ax.set_ylabel('Sepal length')
ax.set_zlabel('Petal length')
ax.set_title('Iris Dataset')
ax.dist = 12
plt.show()
```