Data visualisation in Python

Prof. Dorien Herremans

For a full overview of types of plots using matplotlib, see the gallery at https://matplotlib.org/2.0.2/gallery.html

Scatterplots

We will be using scottish_hills.csv from https://github.com/ourcodingclub/CC-python-pandas-matplotlib. The file contains all the mountains above 3000 feet (about 914 metres) in Scotland.

We can read this into a variable and see the first 10 lines:

In [1]:
```python
import pandas as pd

dataframe = pd.read_csv("scottish_hills.csv")
print(dataframe.head(10))
```

```
                   Hill Name  Height   Latitude  Longitude    Osgrid
0      A' Bhuidheanach Bheag   936.0  56.870342  -4.199001  NN660775
1              A' Chailleach   997.0  57.693800  -5.128715  NH136714
2              A' Chailleach   929.2  57.109564  -4.179285  NH681041
3  A' Chraileag (A' Chralaig)  1120.0  57.184186  -5.154837  NH094147
4            A' Ghlas-bheinn   918.0  57.255090  -5.303687  NH008231
5              A' Mhaighdean   967.0  57.719644  -5.346720  NH007749
6             A' Mharconaich   973.2  56.857002  -4.290668  NN604762
7                 Am Basteir   934.0  57.247931  -6.202982  NG465253
8                  Am Bodach  1031.8  56.741727  -4.983393  NN176650
9              Am Faochagach   953.0  57.771801  -4.853899  NH303793
```

As explored last week, pandas dataframes can be used for some preliminary data exploration. For instance, let's sort the hills by height:

In [2]:
```python
sorted_hills = dataframe.sort_values(by=['Height'], ascending=False)

# Let's have a look at the top 5 to check
print(sorted_hills.head(5))
```

```
                      Hill Name  Height   Latitude  Longitude    Osgrid
92                    Ben Nevis  1344.5  56.796891  -5.003675  NN166712
88   Ben Macdui (Beinn Macduibh)  1309.0  57.070368  -3.669099  NN988989
104                   Braeriach  1296.0  57.078298  -3.728389  NN953999
115                  Cairn Toul  1291.0  57.054397  -3.710773  NN963972
212         Sgor an Lochain Uaine  1258.0  57.058369  -3.725797  NN954976
```

Now let's load matplotlib. Note: if you are using a jupyter notebook you need the inline statement on line 1 below:

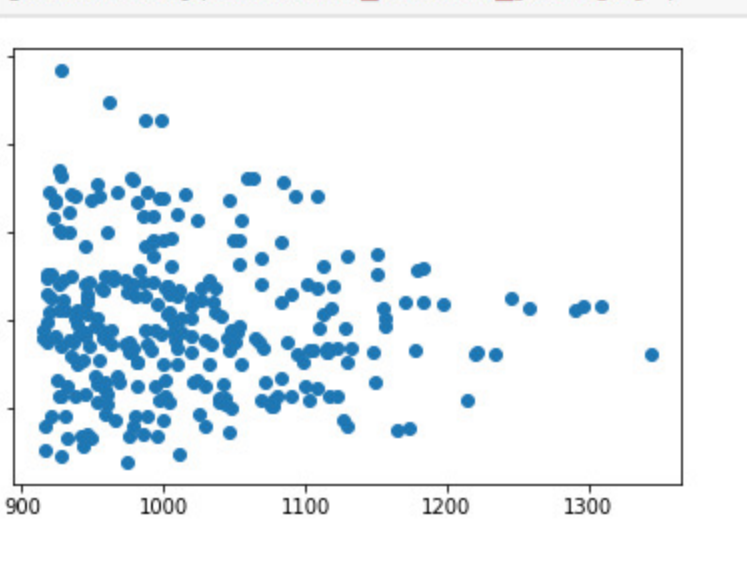
In [3]:
```python
1 %matplotlib inline
2 import matplotlib.pyplot as plt
```

To save us some time, let's create some shortcut variables, x and y, to register the hight and latitude coordinates of each of the hills.

In [4]:
```python
x = dataframe.Height
y = dataframe.Latitude
z = dataframe.Longitude
```

Now we are ready to start visualising them. Let's create (and save) a scatterplot:

In [5]:
```python
plt.scatter(x, y)
plt.savefig("scottish_scatter_plot.png")
```



If you are not using iPython, you can use `plt.show()` to display the plot.

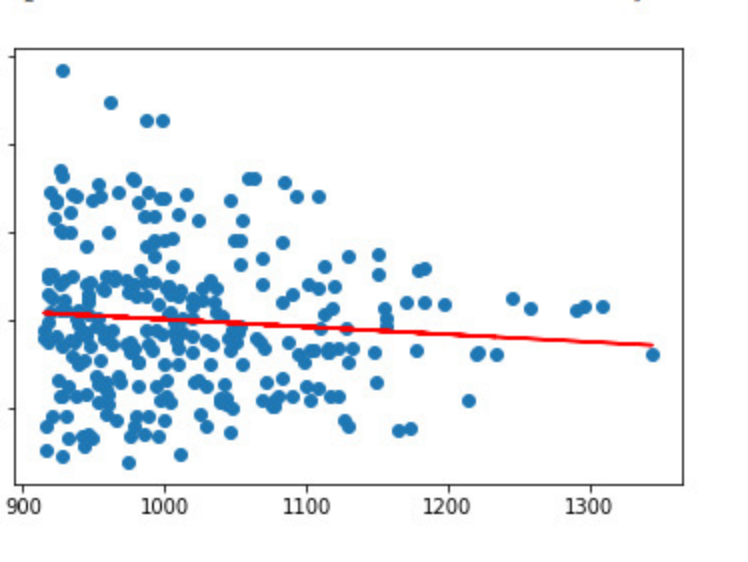
Now let's build upon this graph by adding a linear regression line to it.

In [6]:
```python
from scipy.stats import linregress
stats = linregress(x, y)

m = stats.slope
b = stats.intercept
```

Now we can add the plot of our linear regression by using the equation of a straight line:

In [7]:
```python
plt.scatter(x, y)
plt.plot(x, m * x + b, color="red")   # The equation of the straight line.
```

Out[7]: `[<matplotlib.lines.Line2D at 0x1102d1f98>]`

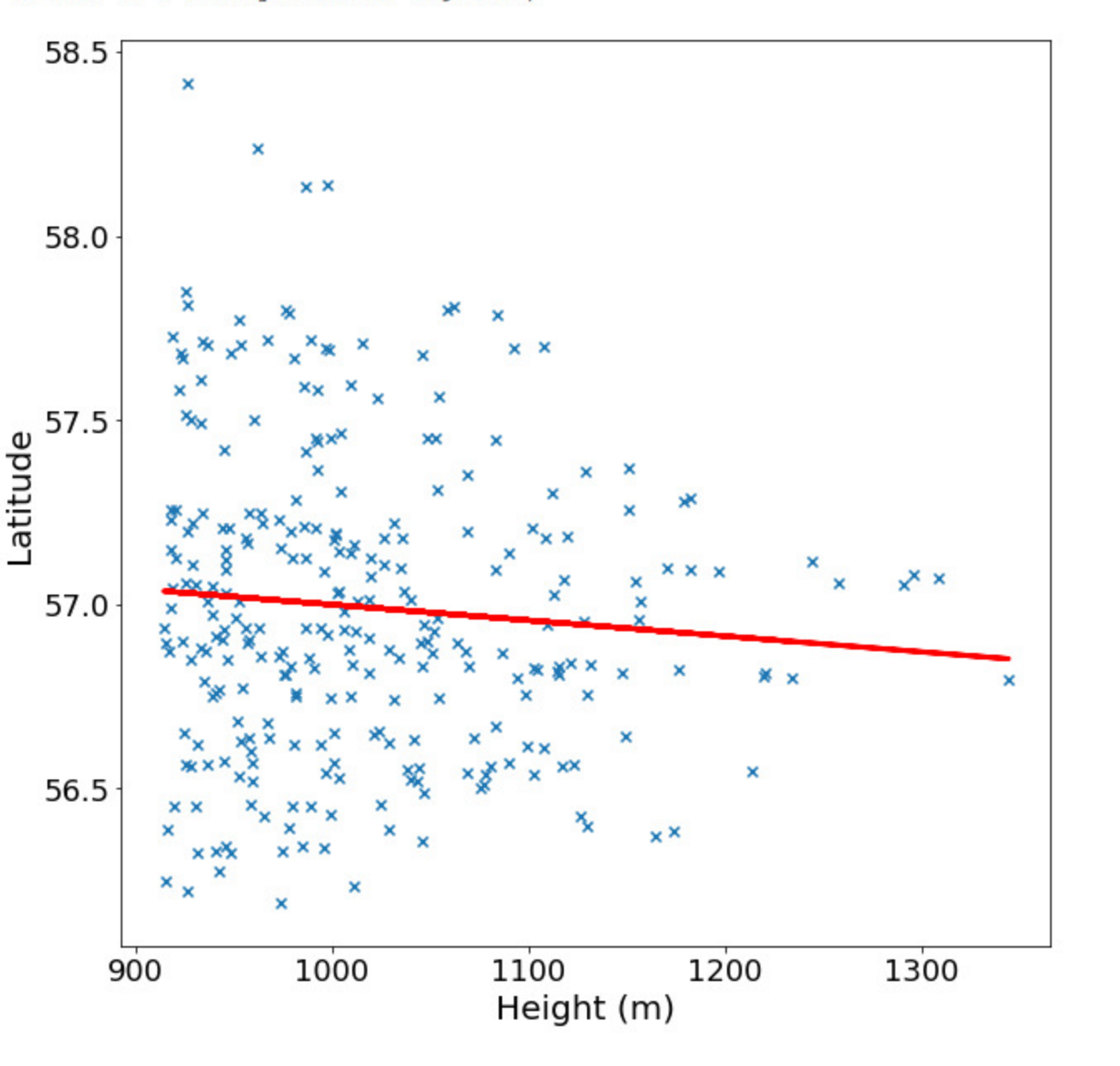


Note, wether this line is statistically significant can be determined using the extra information in the stats object - `stats.rvalue` and `stats.pvalue`.

Now you can make your plot look nicer using arguments such as as fontsize, linewidth, color,...

In [8]:
```python
# Change the default figure size
plt.figure(figsize=(10,10))

# Change the default marker for the scatter from circles to x's
plt.scatter(x, y, marker='x')

# Set the linewidth on the regression line to 3px
plt.plot(x, m * x + b, color="red", linewidth=3)

# Add x and y lables, and set their font size
plt.xlabel("Height (m)", fontsize=20)
plt.ylabel("Latitude", fontsize=20)

# Set the font size of the number lables on the axes
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
```

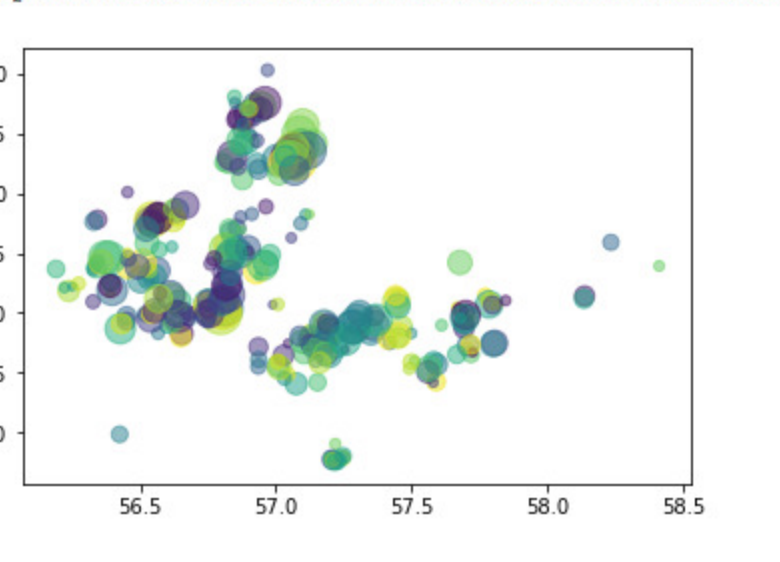
Out[8]:
```
(array([ 56. ,  56.5,  57. ,  57.5,  58. ,  58.5,  59. ]),
 <a list of 7 Text yticklabel objects>)
```



Let's have a look at how the hills our spread out geographically using latitude (y) and longitude (z). Now we can use s = x to say that the size needs to be equal to the height (x). (I added -900 to make the difference between big and small mountains larger)

In [9]:
```python
import numpy as np

colors = np.random.rand(len(y))  #generates a different color for each different mountain

plt.scatter(y, z, s = (x-900), c=colors, alpha=0.5)
```

Out[9]: `<matplotlib.collections.PathCollection at 0x110581a20>`

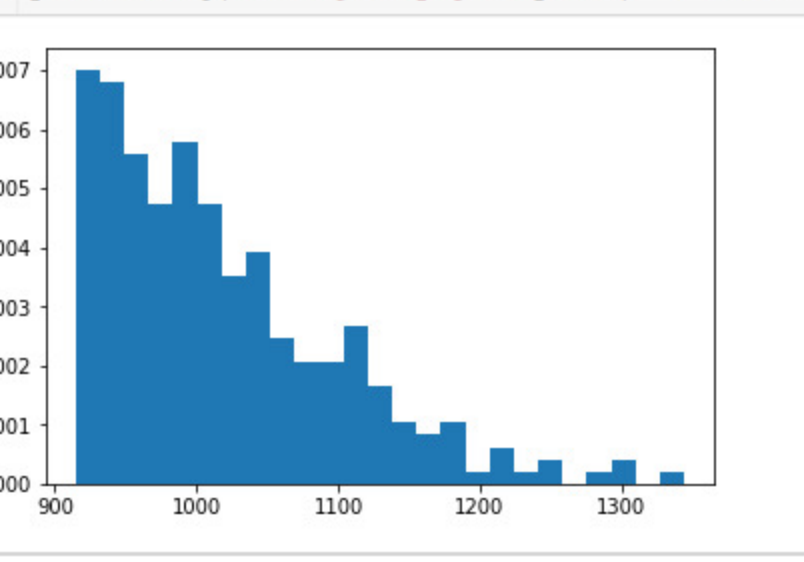


Histograms

Let's try some other graphs. A full selection is given at the matplotlib website.

Let's start by looking at the distribution of our hills over the latitude (variable y from earlier):

In [10]:
```python
plt.hist(x, bins=25, normed=True) #bins separates the latitude in 25 discrete categories. Normed will normalize the

plt.savefig("histogram.png", dpi=25)  # results in 160x120 px image
```

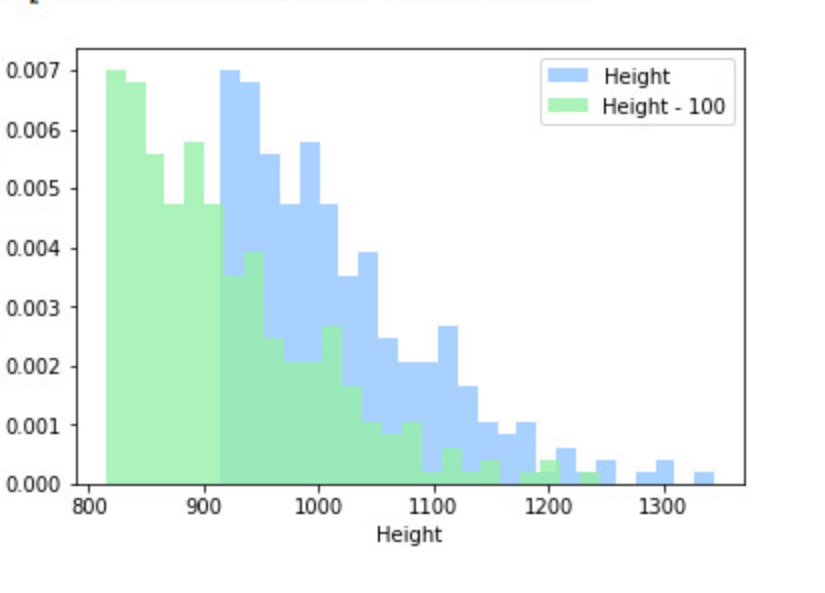


Quickly style your plot with stylesheets, full overview at https://matplotlib.org/gallery/style_sheets/style_sheets_reference.html.

Let's also create a new variable that contains the height of the hills -100m. This to illustrate how to add a second distribution to your graph. In this case, we will make them slightly transparent.

In [11]:
```python
import numpy as np

# using a stylesheet:
plt.style.use('seaborn-pastel')

#creating the new height variable:
shifted_x = x - 100

fig, ax = plt.subplots()
ax.hist(x, bins=25, normed=True, histtype="stepfilled", alpha=0.8,label='Height')
ax.hist(shifted_x, bins=25, normed=True, histtype="stepfilled", alpha=0.8,label='Height - 100')
ax.legend(prop={'size': 10})

ax.set_ylabel('Normalised distribution')
ax.set_xlabel('Height')
```

Out[11]: `<matplotlib.text.Text at 0x11044f8d0>`

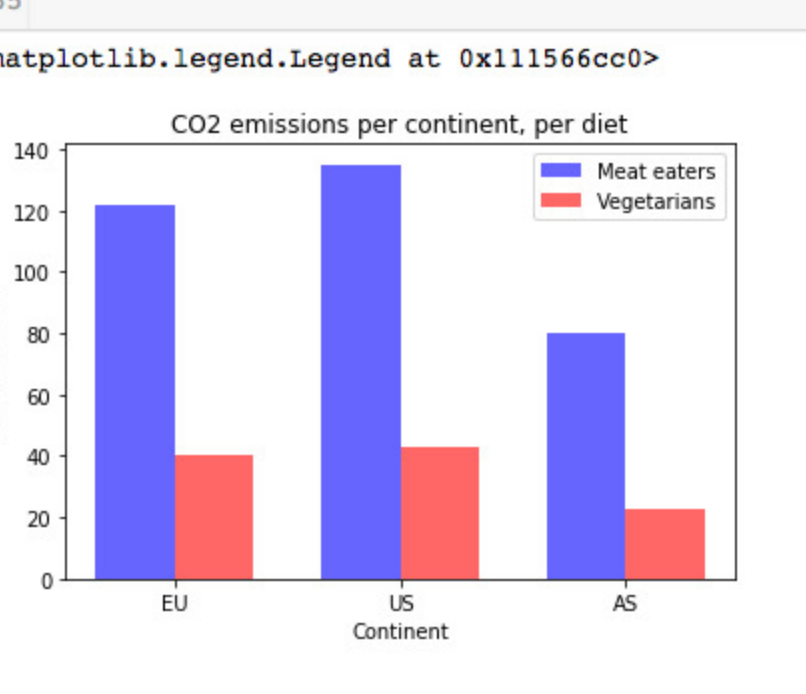


Bar charts

Plot the average CO2 output of both vegetarians and meat eaters for different continents in a bar plot. The data is given below:

In [12]:
```python
labels = ['EU', 'US', 'AS']

meateaters = [122, 135, 80]
vegetarians = [40, 43, 23]


# Keep a numeric index for the x-axis lables. This will be the position sof the ticks.
index = np.arange(len(labels))


fig, ax = plt.subplots()

bar_width = 0.35
opacity = 0.6

rects1 = ax.bar(index, meateaters, bar_width,
                alpha=opacity, color='b',
                label='Meat eaters')

# Note, shift the x postion also with the bar width so that the bar appears next to the previous onet
rects2 = ax.bar(index + bar_width, vegetarians, bar_width,
                alpha=opacity, color='r',
                label='Vegetarians')


ax.set_xlabel('Continent')
ax.set_ylabel('CO2 emissions')
ax.set_title('CO2 emissions per continent, per diet')

ax.set_xticks(index + bar_width / 2)

ax.set_xticklabels(labels)

ax.legend()

```

Out[12]: `<matplotlib.legend.Legend at 0x111566cc0>`



Line plots

Let's move on to another type of graph: a simple line plot, but using two vertical axis.

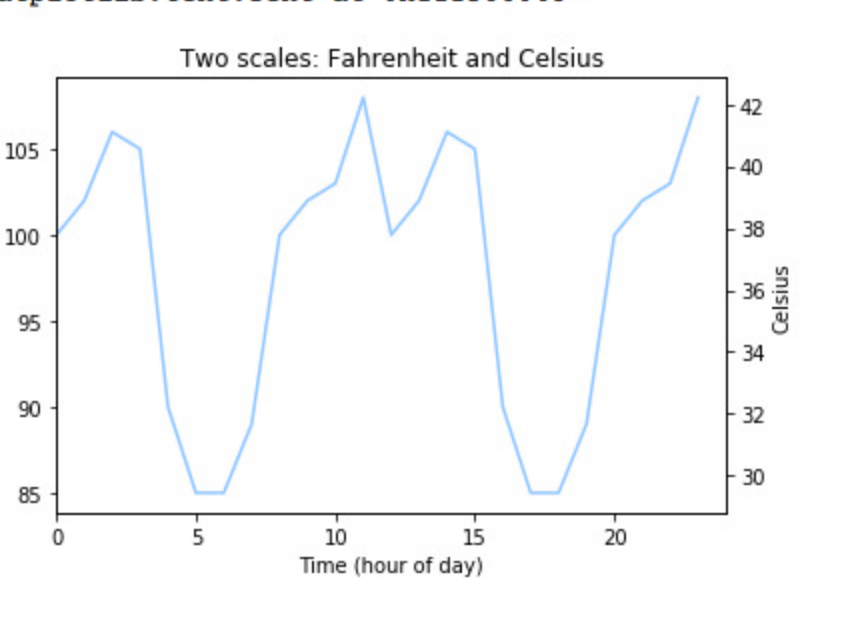
We will create a function to calculate the temperature in celcius given Fahrenheit.

In [13]:
```python
heit2celsius(temp):

s temperature in Celsius.

 (5. / 9.) * (temp - 32)


'URE AT EACH HOUR THROUGHOUT THE DAY (In Fahrenheit)
e = [100, 102, 106, 105, 90, 85, 85, 89, 100, 102, 103, 108, 100, 102, 106, 105, 90, 85, 85, 89, 100, 102, 103, 108]


```

We will use twinx to get a second set of axes. This allows us to plot the temperature evolution throughout the day using both C and F.

In [14]:
```python
fig, ax_f = plt.subplots()
ax_c = ax_f.twinx()

#plot our data:
ax_f.plot(temperature)
ax_f.set_xlim(0, 24) #x-axis shows 24 hours

# set the axis limits:
y1, y2 = ax_f.get_ylim()
ax_c.set_ylim(fahrenheit2celsius(y1), fahrenheit2celsius(y2))
ax_c.figure.canvas.draw()

# change some axis labels
ax_f.set_title('Two scales: Fahrenheit and Celsius')
ax_f.set_ylabel('Fahrenheit')
ax_c.set_ylabel('Celsius')
ax_f.set_xlabel('Time (hour of day)')

```

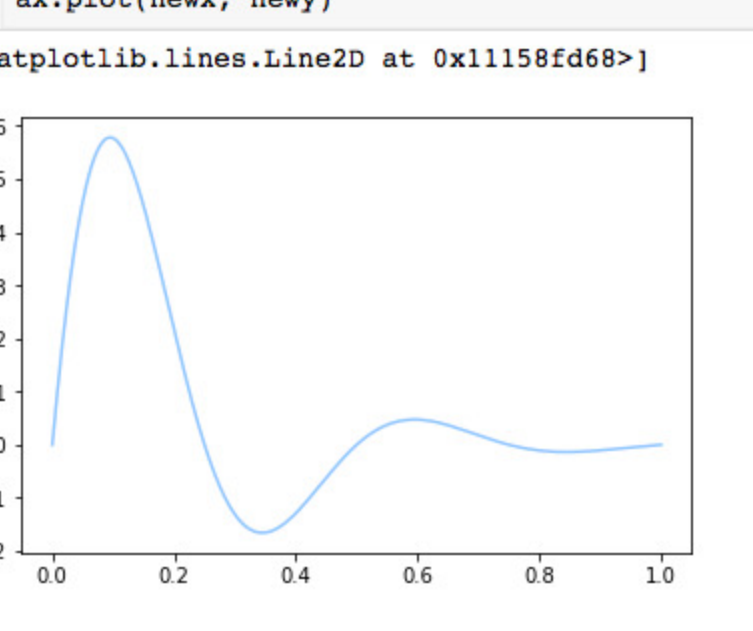
Out[14]: `<matplotlib.text.Text at 0x111566048>`



Suppose we have a sinusoidal plot (formula line 2 below), and we want to fill it. First let's plot the sinusoid.

In [15]:
```python
1 newx = np.linspace(0, 1, 500) # sample 500 X's between 0 and 1
2 newy = np.sin(4 * np.pi * newx) * np.exp(-5 * newx) #formula for our graph
3
4 fig, ax = plt.subplots()  #plt.subplots()  lets us acces the axis and plot seperately.
5
6 ax.plot(newx, newy)
```

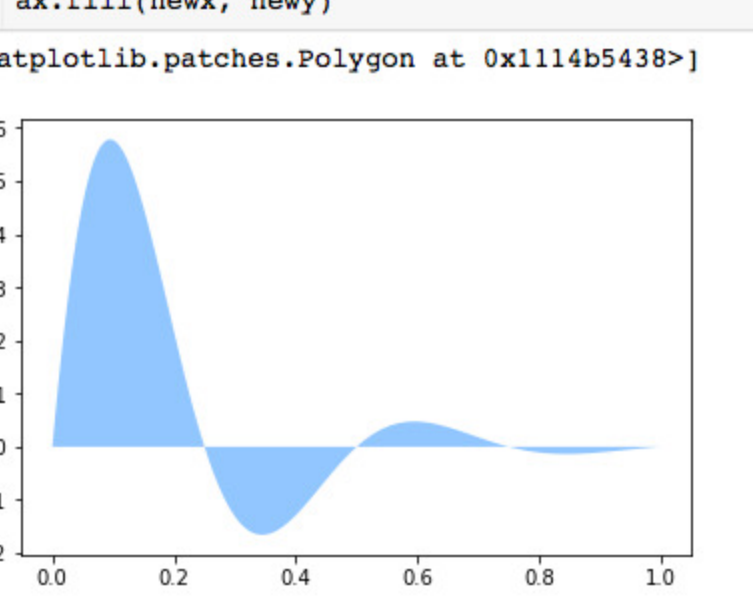
Out[15]: `[<matplotlib.lines.Line2D at 0x11158fd68>]`



Then we can easily use the fill command.

In [16]:
```python
fig, ax = plt.subplots()

ax.fill(newx, newy)
```

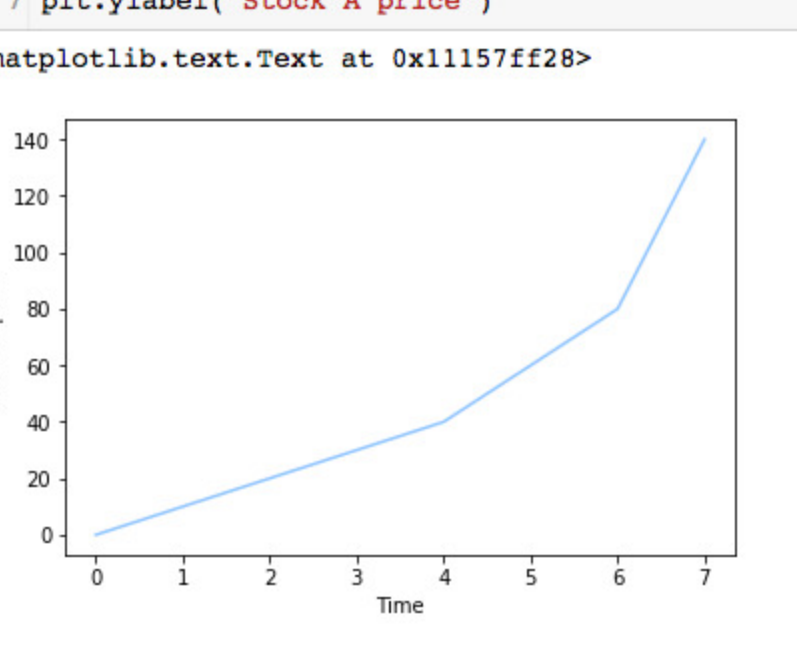
Out[16]: `[<matplotlib.patches.Polygon at 0x1114b5438>]`



Axis labels can influence how we perceive the data. Let's have a look at this stock, which has been stagnating recently.

In [17]:
```python
# growth of stock A
stockA = [0, 10, 20, 30, 40, 60, 80, 140]

plt.plot(stockA)
# plt.ylim(bottom=0, top = 110)
plt.xlabel('Time')
plt.ylabel('Stock A price')
```

Out[17]: `<matplotlib.text.Text at 0x11157ff28>`



Now let's change to a logaritmic axis. Other options here are linear, log, logit, symlog. Give it a try.

fig = plt.figure() ax = fig.add_subplot(1, 1, 1)

line, = ax.plot(stockA) ax.set_yscale('log') #change the scale here

plt.xlabel('Time') plt.ylabel('Stock A price') ax.set_title('Logarithmic')

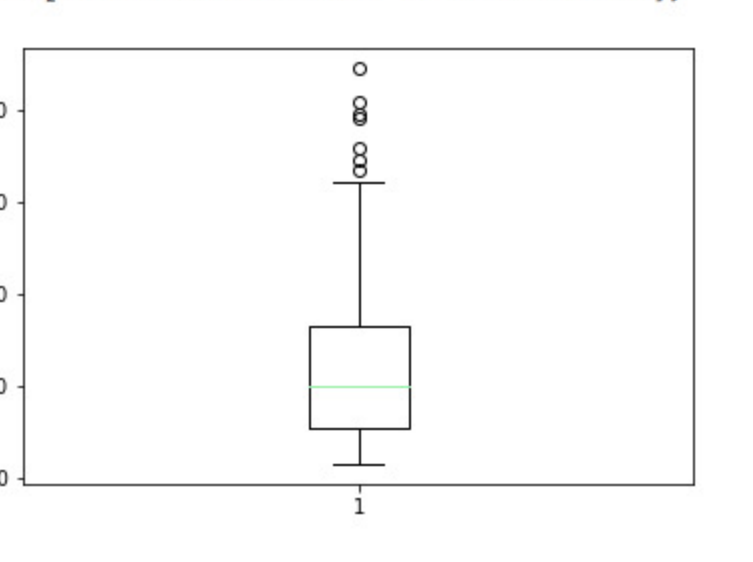
This illustrates how axes can deform the data...

Boxplots

For all the mountains, let's see what their average height is, with standard deviation in a boxplot.

In [18]:
```python
plt.boxplot(x)
```

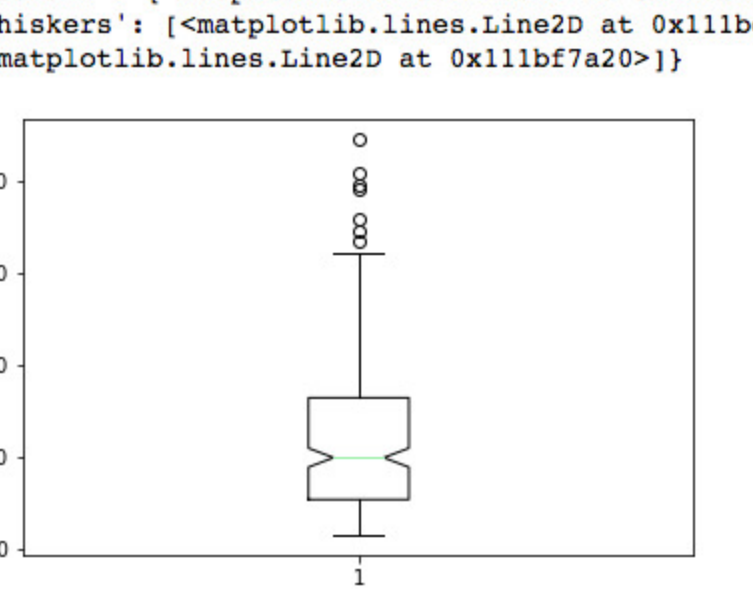
Out[18]:
```
{'boxes': [<matplotlib.lines.Line2D at 0x11107a550>],
 'caps': [<matplotlib.lines.Line2D at 0x11108d6a0>,
  <matplotlib.lines.Line2D at 0x11108d278>],
 'fliers': [<matplotlib.lines.Line2D at 0x111093550>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x111093668>],
 'whiskers': [<matplotlib.lines.Line2D at 0x11107a668>,
  <matplotlib.lines.Line2D at 0x11107a358>]}
```

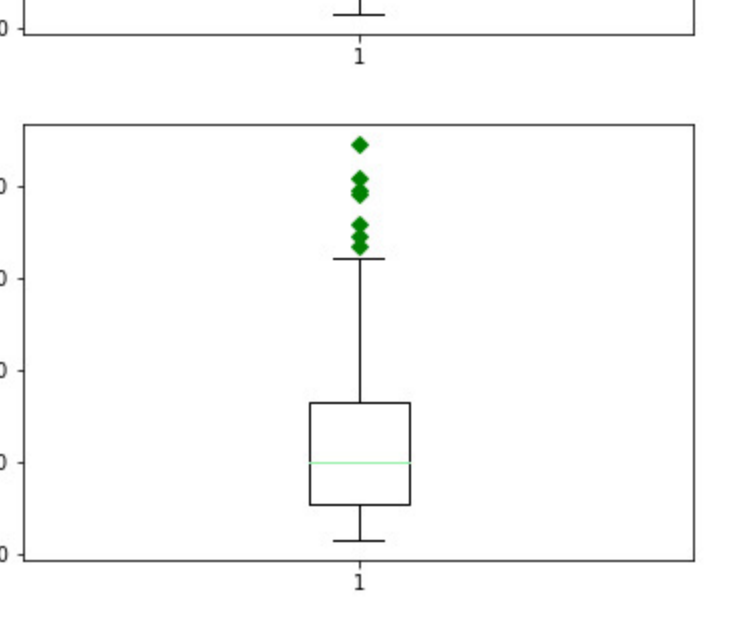


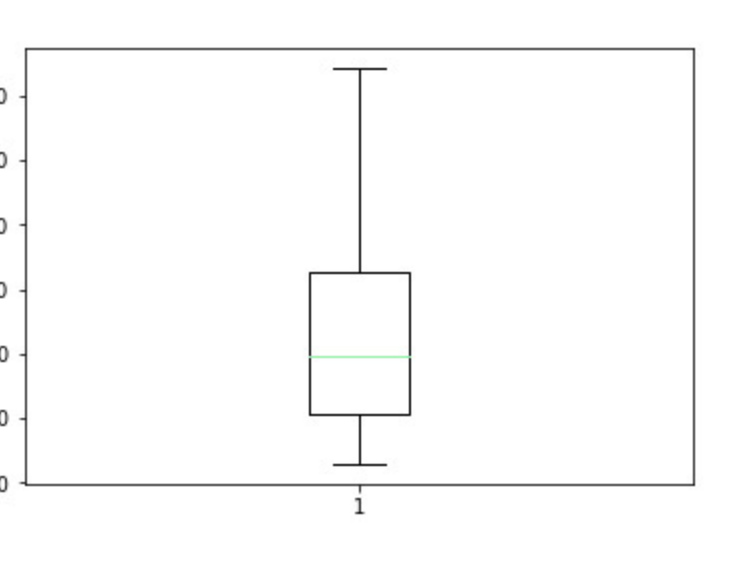
We can make this slightly nicer:

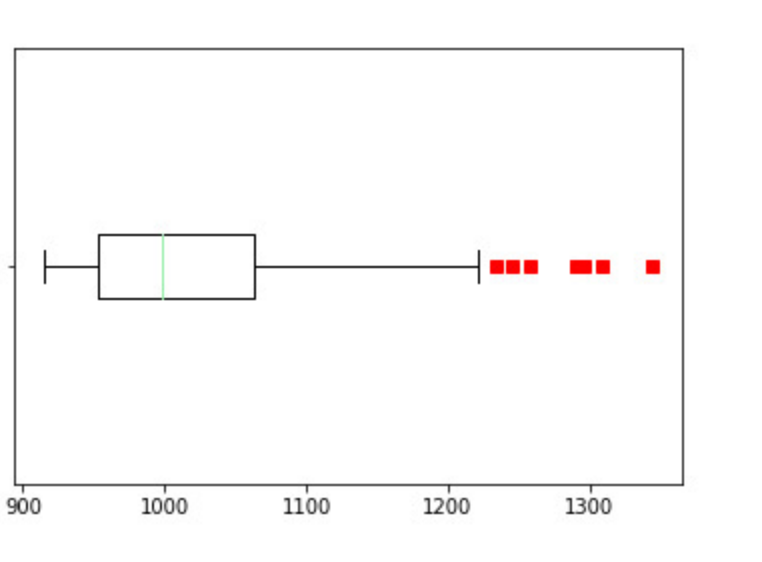
In [19]:
```python
# notched plot
plt.figure()
plt.boxplot(x, 1)

# change outlier point symbols
plt.figure()
plt.boxplot(x, 0, 'gD')

# don't show outlier points
plt.figure()
plt.boxplot(x, 0, '')

# horizontal boxes
plt.figure()
plt.boxplot(x, 0, 'rs', 0)

# change whisker length
plt.figure()
plt.boxplot(x, 0, 'rs', 0, 0.75)
```

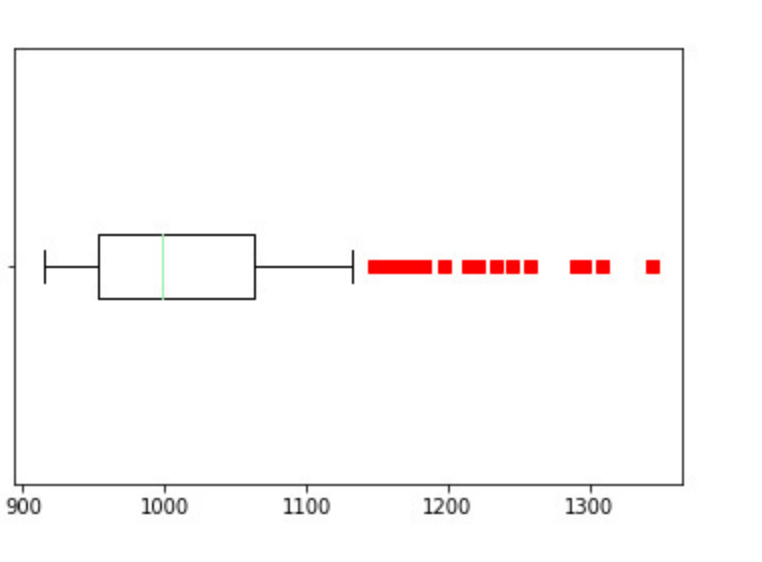
Out[19]:
```
{'boxes': [<matplotlib.lines.Line2D at 0x111bed860>],
 'caps': [<matplotlib.lines.Line2D at 0x111bf7be0>,
  <matplotlib.lines.Line2D at 0x111bfca90>],
 'fliers': [<matplotlib.lines.Line2D at 0x111c06b00>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x111bfcc50>],
 'whiskers': [<matplotlib.lines.Line2D at 0x111beda20>,
  <matplotlib.lines.Line2D at 0x111bf7a20>]}
```











Now to try yourself:

Load the dataset from `https://raw.githubusercontent.com/plotly/datasets/master/school_earnings.csv`, and have a look at what it contains.

Then create the following:

1. A **histogram** of the salaries for women.
2. Add the men's salaries to this histogram.
3. Give your histogram a dark background and label the axes.
4. Next, please label the colors of the histogram so we know who is what (men vs women).
5. Instead of a histogram, create a **bar chart** that lists the salary for women (y-axis) for each school.
6. Also add men to this bar chart.
7. Make the style nice and add labels.
8. Now create a nice **boxplot** of the data, one for men, one for women (two box's same graph).

Solution: histogram

In [ ]:

In [ ]:

Solution: Bar chart

In [ ]:

Solution: Boxplot

In [ ]:

Well done!

In [ ]: