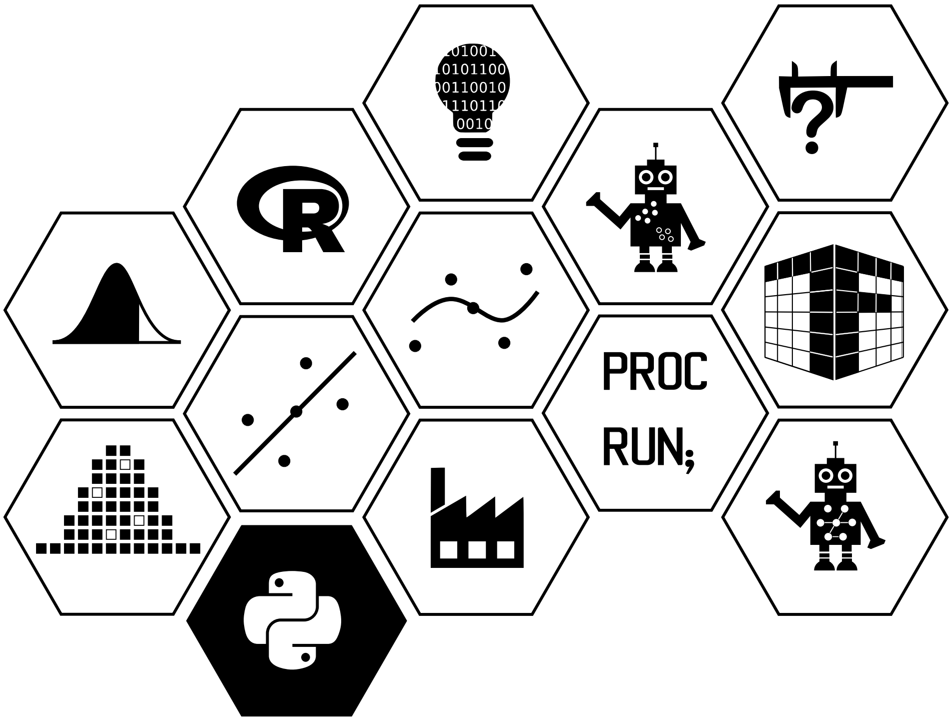
Data Programming in Python

Andrew Elliott and Vinny Davies
Original notes by Chaitanya Kaul and Ludger Evers

Academic Year 2021-22

Week 9:

Plotting in Python



Overview

All the approaches to plotting we will look at this week make use of `matplotlib`, a Python library for 2D plotting. We will however look at different front ends:

- `matplotlib` provides a module called `pyplot`, which provides a MATLAB-inspired interface for plotting. It is however rather clunky and can be cumbersome to use.
- `seaborn` provides a more high-level interface which is similar to the `qplot` function in `ggplot2` in R.
- pandas has some plotting functionality, which can be useful when quickly creating a plot.

Both `seaborn` and `pandas` make use of `matplotlib`.

Gallery for ‘matplotlib‘

https://matplotlib.org/gallery/index.html

Gallery for ‘seaborn‘

https://seaborn.pydata.org/examples/index.html

Another framework worth mentioning is `ggplot` a “port” of `ggplot2` to Python. Unfortunately `ggplot` is not being actively maintained any more.

Interactive web apps involving plots (“Shiny for Python”)

Python has at least two frameworks for showing interactive plots in a way similar to Shiny in R.

- Bokeh
- Plot.ly Dash

Both frameworks are similar in their approach. In both cases, you do not need to know any Javascript to build a web app using the existing components, but writing your own components or changing how components interact requires knowledge of Javascript. Custom components in Dash need to be written in React.

Bokeh has more features, but Dash appears to be coded up more cleanly.

We will unfortunately not be able to cover Bokeh or Dash in this course.

Gallery for Bokeh

https://bokeh.pydata.org/en/latest/docs/gallery.html

Gallery for Dash

https://dash.plot.ly/gallery

matplotlib

matplotlib's pyplot module provides a interface for creating plots, which is heavily inspired by MATLAB.

Creating a first plot with matplotlib

The example below uses the function plot from pyplot to create a line plot.

Example 1.

The code below draws the sine function for $x \in [0, 2\pi]$.

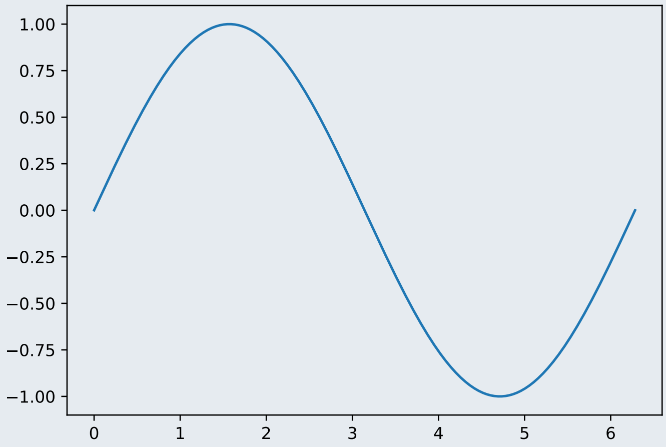
```
import matplotlib.pyplot as plt
import numpy as np
import math

x = np.linspace(0, 2*math.pi, 1000)     # Create a regularly spaced sequence
y = np.sin(x)                           # Calculate sine

plt.figure()                            # Create a new figure
```

```
## <Figure size 650x450 with 0 Axes>
```

```
plt.plot(x, y)                          # Draw line plot
```

```
## [<matplotlib.lines.Line2D object at 0x00000190D1AC2710>]
```

```
plt.show()                              # Show the plot
```



```
plt.close()                             # Close plot and free up resources
```

We have called three functions from pyplot.

- figure creates a new plot. It takes as optional arguments (amongst others):
  - figsize, a tuple of integers giving the width and height of the figure in inches (default 6.4 x 4.8in),
  - dpi, an integer giving the resolution of the figure (default 100dpi), and
  - facecolor, a string giving the background colour

  You do not necessarily have to call figure. Drawing a plot will trigger the creation of a new figure if there is no current figure.

- plot creates a plot. We will look at plot and other functions for creating plots in more detail soon.

- show shows the plot. Depending on how you run Python it may not be necessary to call show. You do not need to call show in Jupyter or in Spyder.

- close closes the plot properly and frees up the memory used by the plot. However, unless you create a large number of plots you do not need to worry about calling close.

plot

In this section we will look at the function `plot` in more detail. `plot` draws a plot of one variable against another, using either points or lines (or both).

We have already seen that plot takes as arguments the horizontal ("x") and vertical ("y") coordinates. If only one positional argument is provided it is interpreted as the vertical coordinate with the zero-based index used as the vertical coordinate.

Customising plots `plot` takes a large number of optional arguments, which can be used to control how the data is plotted. The most important ones are given in the table below.

Optional Argument	Description
`color`	Colour to be used, either a letter (`"r"`, `"g"`, `"b"`, `"c"`, `"y"`, `"m"`, `"k"` (black), `"w"`), an HMTL name (e.g., `"indigo"`) a hex triplet (e.g. `"#ff0000"` for red)
`alpha`	Transparency ranging from 0 (transparent) to 1 (opaque)
`marker`	Marker to be used for points (`"."`, `","`, `"o"`, `"s"`, `"+"`, `"x"`, ...)
`markersize`	Size of the marker
`linestyle`	Line style to be used (`"-"` (solid), `"--"` (dashed), `";"` (dot-dash), `":"` (dotted), use `""` or `"  "` to suppress lines)
`linewidth`	Width of the lines drawn

By default `plot` produces a line plot, but it will also draw points if the argument `marker` is provided (or an equivalent format string). To only show points, set `linestyle=""`.

Note that the optional arguments do not accept vectors and the value provided applies to all data points and lines. In other words, a single call to `plot` can only plot points in one colour using one marker or draw lines in one colour.

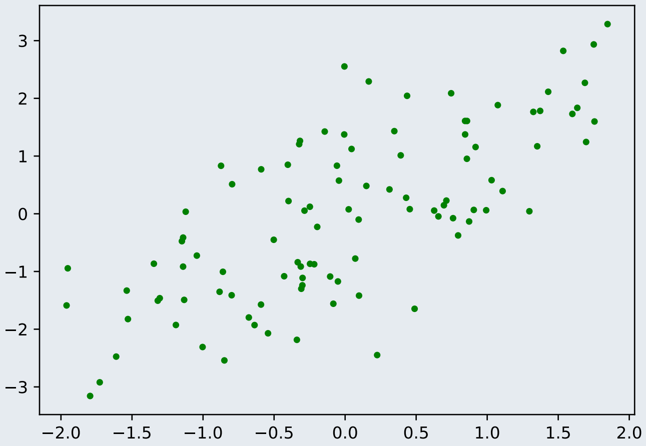
Example 2.

We can produce a scatter plot using green circles as markers using

```
x = np.random.standard_normal(100)      # Create a random data cloud
y = x + np.random.standard_normal(100)
plt.plot(x, y, marker=".", color="g", linestyle="")
                                        # Draw scatter plot
```

```
## [<matplotlib.lines.Line2D object at 0x00000190D75784E0>]
```

```
plt.show()                              # Show plot
```



https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html

The API documentation of `plot` contains the documentation of all the optional arguments of `plot`.

Format strings `plot` also takes an optional argument `fmt` (can also be provided as third positional argument), which provides a more compact way of specifying the plotting format. `fmt` is a string consisting of up to three parts specifying the colour, the marker and the line type. If marker or line type codes are omitted no points or lines are drawn.

Type	Letter	Equivalent argument
Colour	`b`	`color="b"` (blue)
	`g`	`color="g"` (green)
	`r`	`color="r"` (red)
	`c`	`color="c"` (cyan)
	`m`	`color="m"` (magenta)
	`y`	`color="y"` (yellow)
	`k`	`color="k"` (black)
	`w`	`color="w"` (white)
Markers	`.`	`marker="."` (point)
	`,`	`marker=","` (pixel)
	`o`	`marker="o"` (circle)
	`s`	`marker="s"` (square)
	`+`	`marker="+"` (plus)
	`x`	`marker="x"` (cross)
	`...`	`...`
Line	`-`	`linestyle="-"` (solid)
	`--`	`linestyle="--"` (dashed)
	`;`	`linestyle=";"` (dash-dot)
	`:`	`linestyle=":"` (dotted)

In the plotting commands from example 2 we could have used

```
plt.plot(x, y, "g.")
```

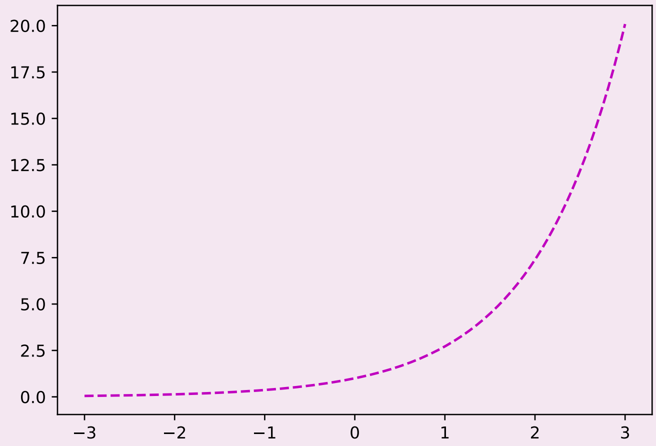
instead of

```
plt.plot(x, y, marker=".", color="g", linestyle="")
```

Task 1.

Create a plot of the exponential function for $x \in [-3, 3]$. Change the colour to magenta and the line to be dashed. The plot should look like the one shown below.

```
## [<matplotlib.lines.Line2D object at 0x00000190D75B1CC0>]
```



https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html

The API documentation of `plot` contains the documentation of all the optional arguments of `plot`.

Adding to plots We can add to an existing plot by simply calling `plot` (or any other plotting function) again, without first calling `figure`. In contrast to R plots, `matplotlib` plots can update the axes so that all the newly plotted objects will fit into the plot.

Example 3.

We can plot a sine function with added noise using the following code.

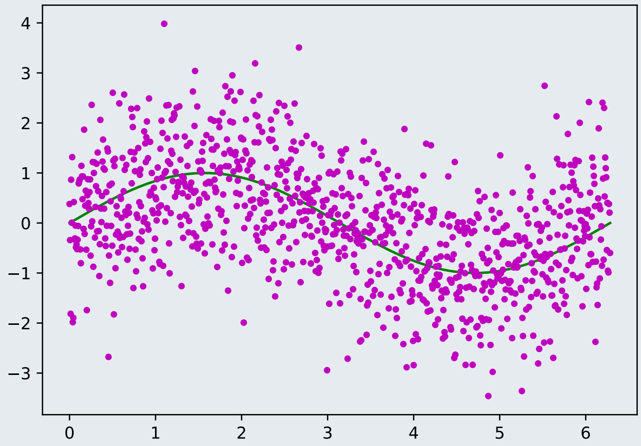
```
x = np.linspace(0, 2*math.pi, 1000)          # Create a regularly spaced sequence
y = np.sin(x)                                # Calculate sine
y_noisy = y + np.random.standard_normal(1000) # Create noisy version of y
plt.plot(x, y, "g")                          # Draw line
```

```
## [<matplotlib.lines.Line2D object at 0x00000190D4D7BF28>]
```

```
plt.plot(x, y_noisy, "m.")                   # Add noisy points
```

```
## [<matplotlib.lines.Line2D object at 0x00000190D4D7BEF0>]
```

```
plt.show()                                   # Show the plot
```



We can actually combine the two calls to `plot` into one call.

```
x = np.linspace(0, 2*math.pi, 1000)          # Create a regularly spaced sequence
y = np.sin(x)                                # Calculate sine
y_noisy = y + np.random.standard_normal(1000) # Create noisy version of y
plt.plot(x, y, "g", x, y_noisy, "m.")        # Draw everything in one go
plt.show()                                   # Show the plot
```

Using the `data` argument If the data is available in a pandas data frame or in a dictionary we can specify the column names to plotted as horizontal and vertical coordinates (as a string) and set the optional argument `data` to the data frame to be used. The basic syntax is

```
plt.plot("columnname_x", "columnname_y", data=dataframe, ...)
```

The example below illustrates the use of the argument `data`.

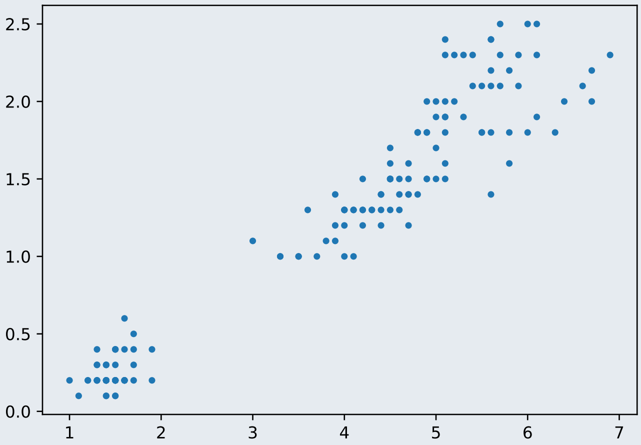
Example 4.

We can plot the petal width against the petal length in the iris data set (from the file `iris.csv` (available from https://github.com/UofGAnalyticsData/DPIP) using the following code.

```
import pandas as pd
iris = pd.read_csv("iris.csv")              # Read in iris data from csv
plt.plot("Petal.Length", "Petal.Width", ".", data=iris)
                                            # Plot from data frame
```

```
## [<matplotlib.lines.Line2D object at 0x00000190D4D7BFD0>]
```

```
plt.show()                                  # Show the plot
```



Most plotting functions accept the `data` argument.

Other types of plots

Scatter plot The function `scatter` draws a scatter plot in which the marker size and colour can be different for each point. The (optional) argument `s` can be used to set the size of the plotting symbol and the (optional) argument `c` can be used to set the colour of the points.

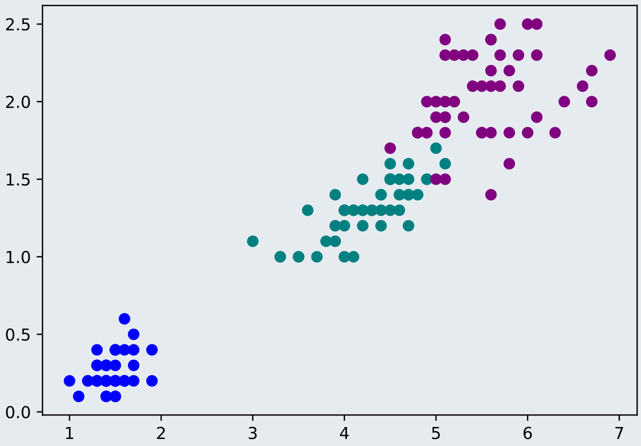
The colour has to either be a valid colour or a numeric value which will be mapped to a colour. Factor-style strings are *not* automatically mapped to colours.

Example 5.

```
colours = iris["Species"].map({"setosa": "blue",  # Manually map species to colours
                                "virginica": "purple",
                                "versicolor": "teal"})
plt.scatter("Petal.Length", "Petal.Width", c=colours, data=iris)
```

```
## <matplotlib.collections.PathCollection object at 0x00000190D7865EB8>
```

```
plt.show()                                   # Show plot
```



https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html

The API documentation of `scatter` contains the documentation of all the optional arguments of `scatter`.

Box plots The function `boxplot` creates a box plot. It takes as argument a vector or list (or tuple) of vectors (in which case a box plot is produced for each vector). The labels of the box plots (shown on the horizontal axis) can be specified as a list using the optional argument `labels`.

Example 6.

We can create a box plot of the distribution of the petal length for the three species in the iris data set in two ways.

One consists of creating a list in which each element is a series containing the petal length measurements for one species.

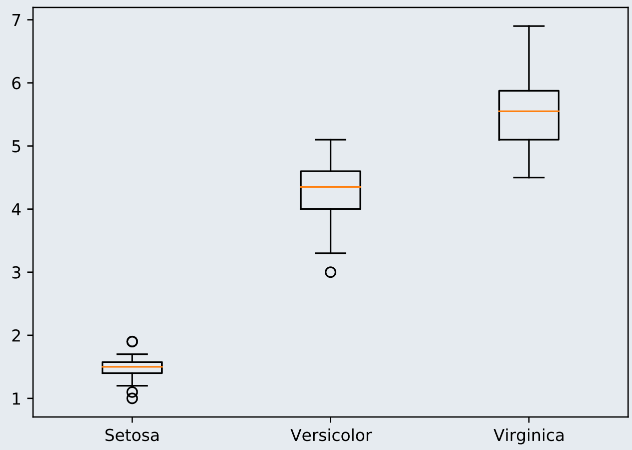
```
pl_by_species = list(map(lambda x:
                         iris.query("Species=='{}'".format(x))["Petal.Length"],
                         ["setosa", "versicolor", "virginica"]))
```

We can then produce the box plots.

```
plt.boxplot(pl_by_species,                        # Create box plots
            labels=["Setosa", "Versicolor", "Virginica"])
```

```
## {'whiskers': [<matplotlib.lines.Line2D object at 0x00000190D75BA898>, <matplotlib.lines.Line
```

```
plt.show()                                        # Show plot
```



Alternatively, we can draw the three box plots one after another. We then have to use the optional argument `positions` to make sure the box plots are not drawn on top of each other.

```
plt.boxplot(iris.query("Species=='setosa'")["Petal.Length"].values,
            positions=[1])
                                                  # Create first box plot
```

```
## {'whiskers': [<matplotlib.lines.Line2D object at 0x00000190D1709DD8>, <matplotlib.lines.Line
```

```
plt.boxplot(iris.query("Species=='versicolor'")["Petal.Length"].values,
            positions=[2])
                                                  # Create second box plot
```

```
## {'whiskers': [<matplotlib.lines.Line2D object at 0x00000190D16E69B0>, <matplotlib.lines.Line
```

```
plt.boxplot(iris.query("Species=='virginica'")["Petal.Length"].values,
            positions=[3])
                                                  # Create third box plot
```

```
## {'whiskers': [<matplotlib.lines.Line2D object at 0x00000190D79E4278>, <matplotlib.lines.Line
```

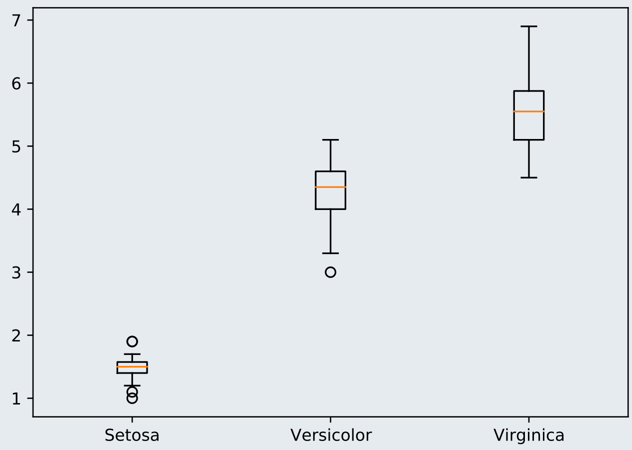
```
plt.xticks([1,2,3], ["Setosa", "Versicolor", "Virginica"])
                                                  # Set correct labels
```

```
## ([<matplotlib.axis.XTick object at 0x00000190D4D40B70>, <matplotlib.axis.XTick object at 0x0
```

```
plt.xlim(0.5,3.5)                                 # Fix horizontal range
```

```
## (0.5, 3.5)
```

```
plt.show()                                        # Show plot
```



Note that we have used the function `xticks` to set the correct labels for the box plots and `xlim` to make sure the range of the horizontal axis is correct.

https://matplotlib.org/api/_as_gen/matplotlib.pyplot.boxplot.html

The API documentation of `boxplot` contains the documentation of all the optional arguments of `boxplot`.

A related function is `violinplot`, which produces a violin plot, a variant of a box plot incorporating a density estimate.

Task 2.

You can simulate three vectors of length 100 from the $N(0, 1)$, $t(5)$ and $t(2)$ distributions using

```
x1 = np.random.standard_normal(100)
x2 = np.random.standard_t(5, 100)
x3 = np.random.standard_t(2, 100)
```

Create box plots of the three columns.

Histograms The function `hist` produces a histogram of the data provided as argument. If the data provided is a vector a single histogram is drawn. If the data provided is a list of vectors, histograms are drawn for each vector. Histograms for more than one vector are drawn as bar charts next to each other, but will be overlaid if the optional argument `histtype="step"` or `histtype="stepfilled"` is used.

Example 7.

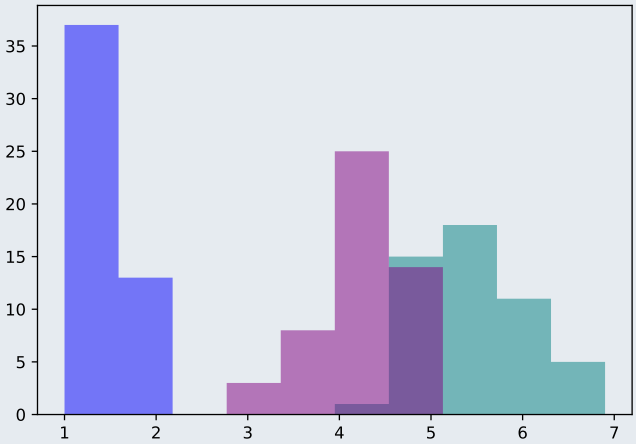
The code below produces a histogram showcasing the differences between the distribution of the petal lengths between the three species.

We can either provide the data as a list of the petal lengths for the three different species from …

```
plt.hist(pl_by_species, color=["blue", "purple", "teal"], alpha=0.5,
         histtype="stepfilled")
```

```
## (array([[37., 13.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
##         [ 0.,  0.,  0.,  3.,  8., 25., 14.,  0.,  0.,  0.],
##         [ 0.,  0.,  0.,  0.,  0.,  1., 15., 18., 11.,  5.]]), array([1.  , 1.59, 2.18, 2.77,
```

```
plt.show()
```



… or call `hist` three times …

```
plt.hist(iris.query("Species=='setosa'")["Petal.Length"],  color="blue", alpha=0.5)
```

```
## (array([ 1.,  1.,  2.,  7., 13., 13.,  7.,  4.,  0.,  2.]), array([1.  , 1.09, 1.18, 1.27, 1
```

```
plt.hist(iris.query("Species=='versicolor'")["Petal.Length"],
         color="purple", alpha=0.5)
```

```
## (array([ 1.,  2.,  3.,  2.,  8.,  7.,  6., 10.,  7.,  4.]), array([3.  , 3.21, 3.42, 3.63, 3
```

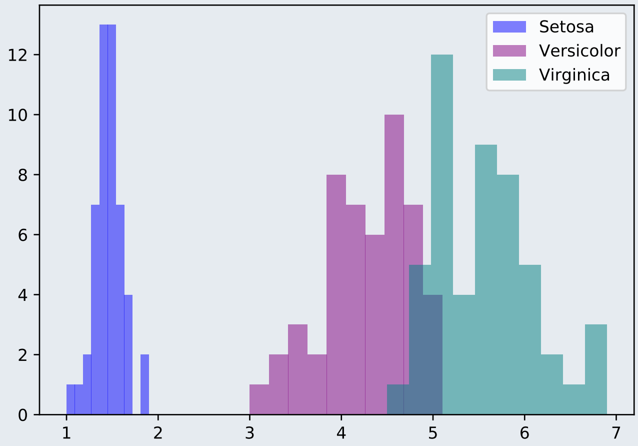
```
plt.hist(iris.query("Species=='virginica'")["Petal.Length"],
         color="teal", alpha=0.5)
```

```
## (array([ 1.,  5., 12.,  4.,  9.,  8.,  5.,  2.,  1.,  3.]), array([4.5 , 4.74, 4.98, 5.22, 5
```

```
plt.legend(labels = ["Setosa", "Versicolor", "Virginica"])
```

```
## <matplotlib.legend.Legend object at 0x00000190D31AD668>
```

```
plt.show()
```



The results are different because the bin width is chosen based on all the data in the former case and only based on each species separately in the latter case.

https://matplotlib.org/api/_as_gen/matplotlib.pyplot.hist.html

The API documentation of `hist` contains the documentation of all the optional arguments of `hist`.

Bar charts and pie charts Bar charts can be drawn using the function `bar` and pie charts can be drawn using the function `pie`. The example below illustrates the functions.

Example 8.

In this example we draw a bar chart and pie chart showing the distribution of the three species in the iris data set (there turn out to be 50 observations for each species).

We start by producing the counts:

```
counts = iris.groupby("Species").apply(lambda x: x.shape[0])
```

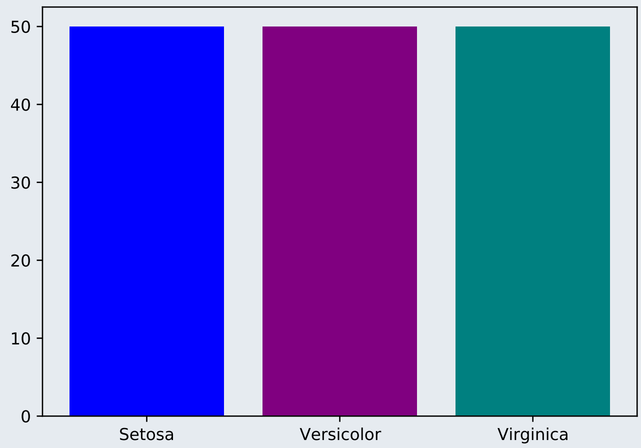
We can then draw a bar chart. `bar` requires the specification of both horizontal positions and heights. We will set the former to `[1,2,3]` and then replace the numerical horizontal axis by a categorical one.

```
plt.bar([1,2,3], counts, color=["blue", "purple", "teal"])
```

```
## <BarContainer object of 3 artists>
```

```
plt.xticks([1,2,3], ["Setosa", "Versicolor", "Virginica"])
```

```
## ([<matplotlib.axis.XTick object at 0x00000190D7814518>, <matplotlib.axis.XTick object at 0x0
plt.show()
```



Alternatively we can draw a pie chart.

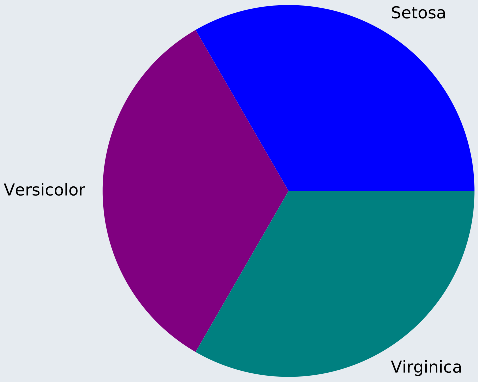
```
plt.pie(counts, colors=["blue", "purple", "teal"],
                labels=["Setosa", "Versicolor", "Virginica"])
```

```
## ([<matplotlib.patches.Wedge object at 0x00000190D7F6FCF8>, <matplotlib.patches.Wedge object
```

```
plt.axis('equal')        # Force equal scales, so that circle looks like a circle
```

```
## (-1.100000018898756, 1.1000000353324482, -1.100000262939185, 1.1000000098602216)
```

```
plt.show()
```



https://matplotlib.org/api/_as_gen/matplotlib.pyplot.bar.html

The API documentation of `bar` contains the documentation of all the optional arguments of `bar`.

https://matplotlib.org/api/_as_gen/matplotlib.pyplot.pie.html

The API documentation of `pie` contains the documentation of all the optional arguments of `pie`.

Plots of functions of two inputs ("3D-style" plots) The 3D-style plotting functions take their input in the form of two optional vectors giving the horizontal and vertical coordinates and a matrix containing the values to be plotted (i.e. they work like the 3D-style plotting functions in R such as `image`, but not like `ggplot2`).

The functions are

```
contour(x, y, z, ...)
contourf(x, y, z, ....)
```

The most important additional arguments are the number of levels to be shown in the plot and the colour map to be used.

The function `colorbar` can used to a legend for the fill colours used.

Example 9.

We will visualise the function

$$f(x, y) = x \cdot \exp(-x^2 - y^2)$$

We start by creating vectors `x` and `y` the matrix `Z` with the function values. We will exploit NumPy's broadcasting rules to create `Z`.

```
x = np.linspace(-2, 2, 100)                    # Create sequence of x's
y = np.linspace(-2, 2, 100)                    # Create sequence of y's

Z = x[None,:] * np.exp(-x[None,:]**2 - y[:,None]**2)
```

We could have also used the NumPy function `meshgrid` to create `Z`

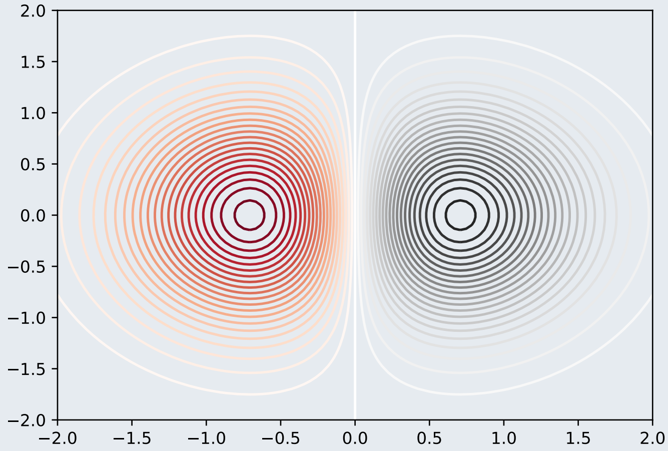
```
X,Y = np.meshgrid(x, y)                        # Expand x and y into matrices
                                               # x row-wise and y column-wise
Z = X * np.exp(-X**2 - Y**2)                   # Define Z in terms of matrices
```

We can now draw a contour plot and a filled contour plot.

```
plt.contour(x, y, Z, 50, cmap='RdGy')          # Draw contour lines
```

```
## <matplotlib.contour.QuadContourSet object at 0x00000190D773AB00>
```

```
plt.show()                                     # Show plot
```



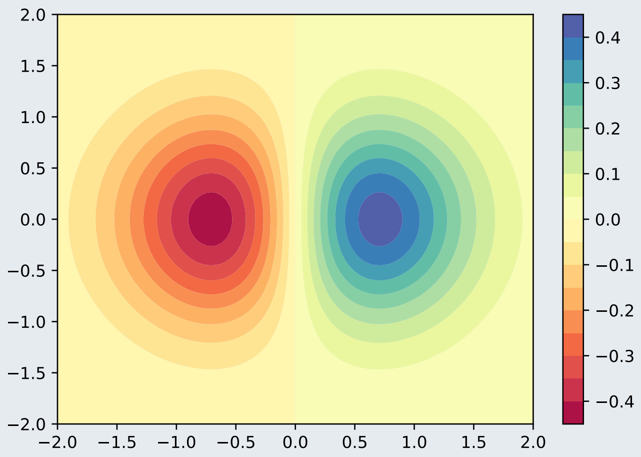
```
plt.contourf(x, y, Z, 20, cmap='Spectral')     # Add contour lines
```

```
## <matplotlib.contour.QuadContourSet object at 0x00000190D7B67710>
```

```
plt.colorbar()                                 # Add a colour bar as legend
```

```
## <matplotlib.colorbar.Colorbar object at 0x00000190D8472BE0>
```

```
plt.show()                                     # Show plot
```

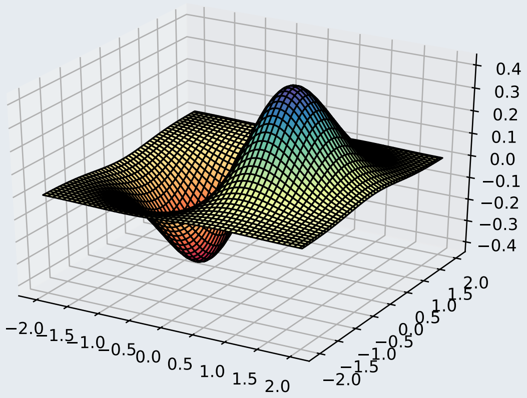


We will finally create a 3D plot using the `mplot3d` toolkit.

```
from mpl_toolkits.mplot3d import Axes3D      # Load required module
ax = plt.axes(projection="3d")               # Set up plot to be 3d
ax.plot_surface(X, Y, Z, cmap="Spectral", edgecolor="black")
                                             # Draw surface plot
```

```
## <mpl_toolkits.mplot3d.art3d.Poly3DCollection object at 0x00000190D8466128>
```

```
plt.show()                                   # Show plot
```



Note that `plot_surface` uses the “expanded” versions `X` and `Y` of `x` and `y`.

https://matplotlib.org/api/_as_gen/matplotlib.pyplot.contour.html

The API documentation of `bar` contains the documentation of all the optional arguments of `contour`.

https://matplotlib.org/api/_as_gen/matplotlib.contourf.bar.html

The API documentation of `bar` contains the documentation of all the optional arguments of `bar`.

mplot3d

https://matplotlib.org/mpl_toolkits/mplot3d/contourf.html

The `mplot3d` toolkit allows for creating 3D plots in matplotlib.

Other functions The `pyplot` interface has many more functions than the ones covered in this week. A detailed list is available in the API documentation.

Pyplot function reference

https://matplotlib.org/api/pyplot_summary.html

Modifying plots

Title The title of a plot can be set using the function `title`, which takes the title for the plot as argument.

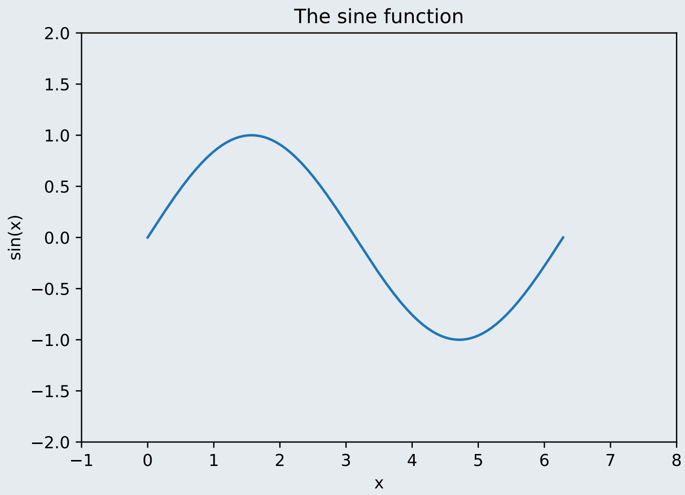
Axis range We can change the range of the axes using the functions `xlim` and `ylim`, which take two numbers (lower and upper end) as their two arguments.

Axis labels The functions `xlabel` and `ylabel` can be used to set the axis labels.

Example 10.

This example illustrates the use of `title`, `xlim`, `ylim`, `xlabel`, and `ylabel`.

```
## <Figure size 650x450 with 0 Axes>

x = np.linspace(0, 2*math.pi, 1000)          # Create sequence from 0 to 2*pi
plt.plot(x, np.sin(x))                       # Draw plot

## [<matplotlib.lines.Line2D object at 0x00000190D8710A20>]

plt.xlim(-1, 8)                              # Set limits of horizontal axis

## (-1.0, 8.0)

plt.ylim(-2, 2)                              # Set limits of vertical axis

## (-2.0, 2.0)

plt.xlabel("x")                              # Set horizontal axis label

## Text(0.5, 0, 'x')

plt.ylabel("sin(x)")                         # Set vertical axis label

## Text(0, 0.5, 'sin(x)')

plt.title("The sine function")               # Set plot title

## Text(0.5, 1.0, 'The sine function')

plt.show()                                   # Show the plot
```



Vertical and horizontal lines Vertical and horizontal lines can be added using the functions `axhline` and `axvline`.

Example 11.

This example illustrates how to add vertical or horizontal lines.

```
## <Figure size 650x450 with 0 Axes>
```

```
x = np.linspace(0, 2*math.pi, 1000)               # Create sequence from 0 tp 2*pi
plt.plot(x, np.sin(x))                            # Draw plot
```

```
## [<matplotlib.lines.Line2D object at 0x00000190D845DD30>]
```

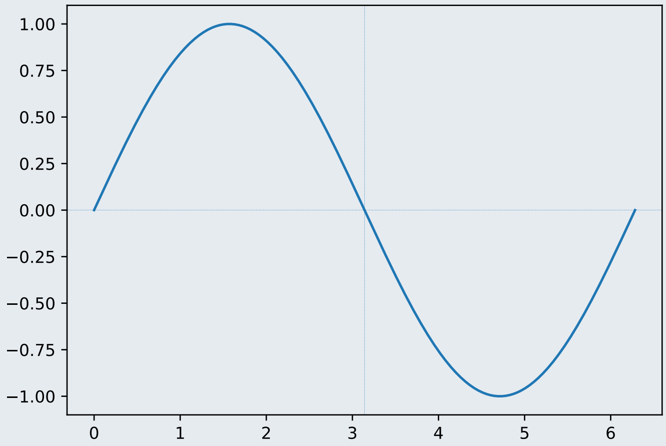
```
plt.axhline(0, linestyle=":", linewidth=0.1)       # Horizontal line at 0
```

```
## <matplotlib.lines.Line2D object at 0x00000190D3C13470>
```

```
plt.axvline(math.pi, linestyle=":", linewidth=0.1) # Vertical line at pi
```

```
## <matplotlib.lines.Line2D object at 0x00000190B228FCC0>
```

```
plt.show()                                        # Show the plot
```



Adding legends Legends can be added to a plot using the function `legend`. `legend` will draw a legend based on the objects already drawn in the plot. The labels can either be provided when drawing the objects using the optional argument `label` or as a tuple as optional argument `labels` when calling `legend`.

Example 12.

In this example we will draw the sine and the cosine function.

```
## <Figure size 650x450 with 0 Axes>
```

```
x = np.linspace(0, 2*math.pi, 1000)            # Create sequence from 0 t 2*pi
plt.plot(x, np.sin(x), "g-", label="sin(x)") # Draw plot of sine function
```

```
## [<matplotlib.lines.Line2D object at 0x00000190D845D7B8>]
```

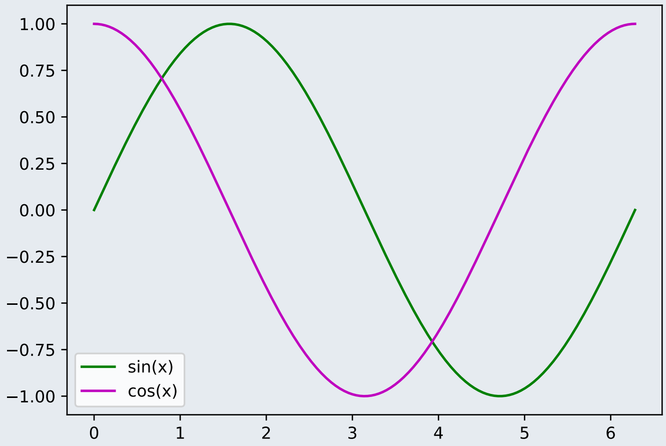
```
plt.plot(x, np.cos(x), "m-", label="cos(x)") # Draw plot of cosine function
```

```
## [<matplotlib.lines.Line2D object at 0x00000190D7F4FB00>]
```

```
plt.legend()                                 # Add a legend
```

```
## <matplotlib.legend.Legend object at 0x00000190D7768F98>
```

```
plt.show()                                   # Show the plot
```



Supplementary material: Themeing plots

matplotlib plots are highly customisable and support themes (“styles”). The easiest way of changing how a plot looks like it to use another style as illustrated by the example below.

```
## <Figure size 650x450 with 0 Axes>
```

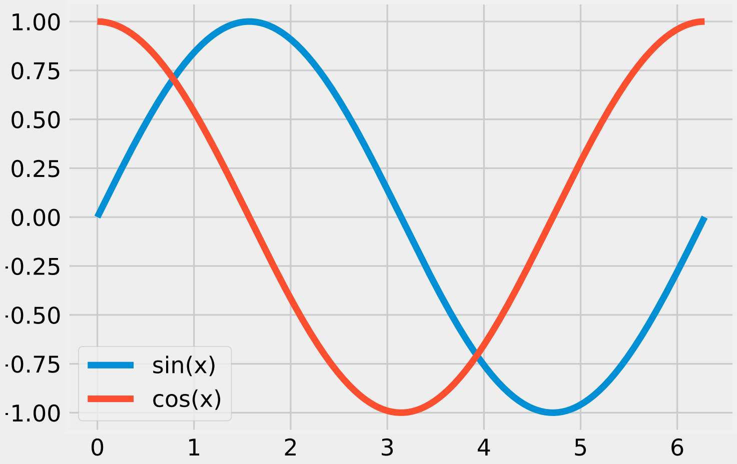
```
plt.style.use('fivethirtyeight')              # Use theme fivethirtyeight
x = np.linspace(0, 2*math.pi, 1000)           # Create sequence from 0 to 2*pi
plt.plot(x, np.sin(x), x, np.cos(x))          # Draw both lines in one go
```

```
## [<matplotlib.lines.Line2D object at 0x00000190D1AB0710>, <matplotlib.lines.Line2D object at 0x00000190D
```

```
plt.legend(labels=["sin(x)", "cos(x)"])       # Add a legend
```

```
## <matplotlib.legend.Legend object at 0x00000190D77F85C0>
```

```
plt.show()                                    # Show the plot
```



A list of styles and a preview of how plots look like for the different styles is available at

https://matplotlib.org/gallery/style_sheets/style_sheets_reference.html

Details of how plots can be customised are available at

https://matplotlib.org/tutorials/introductory/customizing.html

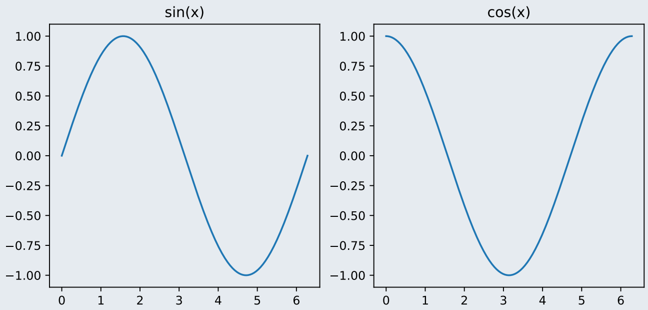
Arranging sub-plots

The function `subplot(nrow, ncol, idx)` creates a sub-plot in a grid of `nrow` × `ncol` plots in position `idx` (one-based).

Example 13.

We can plot a sine curve next to a cosine curve using the following code.

```
plt.figure(figsize=[9,4])              # Create new figure (wider than normal)
## <Figure size 900x400 with 0 Axes>
x = np.linspace(0, 2*math.pi, 1000)    # Create a regularly spaced sequence
plt.subplot(1, 2, 1)                   # Use first plot in row of two plots
## <matplotlib.axes._subplots.AxesSubplot object at 0x00000190D4D27B38>
plt.plot(x, np.sin(x))                 # Plot sine function
## [<matplotlib.lines.Line2D object at 0x00000190D844E470>]
plt.title("sin(x)")                    # Add title
## Text(0.5, 1.0, 'sin(x)')
plt.subplot(1, 2, 2)                   # Use second plot in row of two plots
## <matplotlib.axes._subplots.AxesSubplot object at 0x00000190D7568A20>
plt.plot(x, np.cos(x))                 # Plot cosine function
## [<matplotlib.lines.Line2D object at 0x00000190D7B62F60>]
plt.title("cos(x)")                    # Add title
## Text(0.5, 1.0, 'cos(x)')
plt.show()                             # Show plot
```



Using the object-oriented interface The object-oriented interface for plotting in `pyplot` provides more flexibility for subplots.

Before we look at subplots, we take a quick look at the object oriented interface.

So far we have done all the plotting using functions from `matplotlib.pyplot` which modify the current plot.

However every function has a method of the class `Axes` which is an (almost) identical twin (usually with the same name).

The `subplots` function (with an `s` at the end) provides a much more flexible way of generating subplots that the `subplot` function (without an `s` at the end). Its arguments are given in the table below.

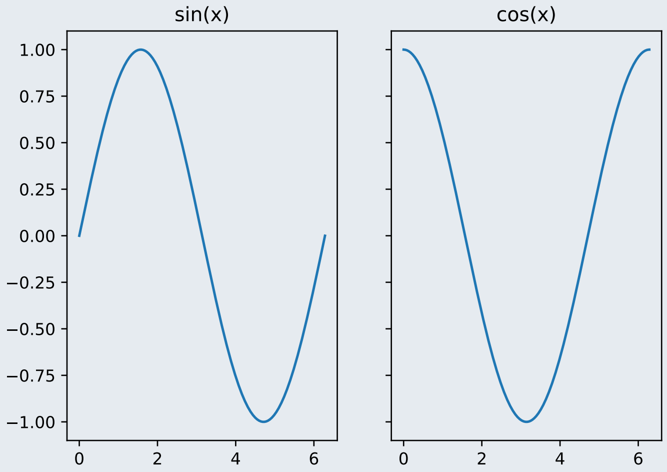
Argument	Description
`nrows`	Number of rows of sub plots (default `1`)
`ncols`	Number of columns of sub plots (default `1`)
`sharex`	Should sub plots share the same x axis (default `False`)
`sharey`	Should sub plots share the same y axis (default `False`)

subplots returns a tuple of an object of the class `Figure` and another tuple of `Axes` objects, which can be used for drawing each sub plot.

Example 14.

We will now rewrite the code from example 13 using `subplots`.

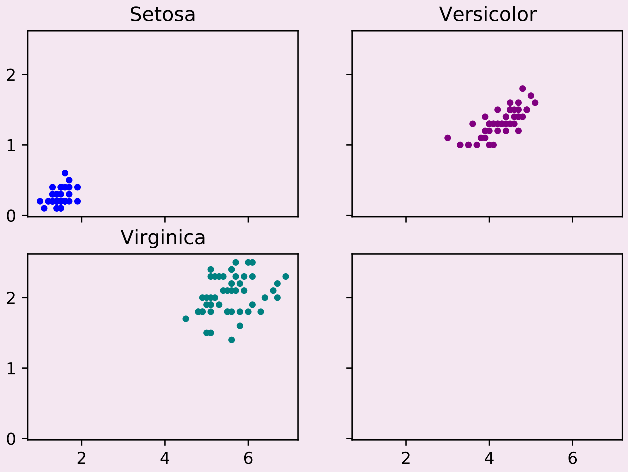
```
x = np.linspace(0, 2*math.pi, 1000)      # Create a regularly spaced sequence
fig, (ax1, ax2) = plt.subplots(1, 2, sharey=True)
                                         # Split the plotting region and share y axis
ax1.plot(x, np.sin(x))                   # Plot sine function

## [<matplotlib.lines.Line2D object at 0x00000190D7BB4F28>]

ax1.set_title("sin(x)")                  # Add title

## Text(0.5, 1.0, 'sin(x)')

ax2.plot(x, np.cos(x))                   # Plot cosine function

## [<matplotlib.lines.Line2D object at 0x00000190D7557DA0>]

ax2.set_title("cos(x)")                  # Add title

## Text(0.5, 1.0, 'cos(x)')

plt.show()                               # Show plot
```



Task 3.

Create a plot of the petal width against the petal length for the iris data, showing the three species in separate subplots using common axes. Your plot should look like the one shown below.

```
## [<matplotlib.lines.Line2D object at 0x00000190D82D0CC0>]
## Text(0.5, 1.0, 'Setosa')
## [<matplotlib.lines.Line2D object at 0x00000190D82D00B8>]
## Text(0.5, 1.0, 'Versicolor')
## [<matplotlib.lines.Line2D object at 0x00000190D776A630>]
## Text(0.5, 1.0, 'Virginica')
```



Saving plots

Plots can be saved to different formats (PNG images; PS, EPS or PDF documents; SVG graphics) using the function `savefig`. The file format will be determined by the extension used in the file name (unless specified otherwise).

Example 15.

The code below illustrates how to save a plot to a PDF file.

```
## <Figure size 650x450 with 0 Axes>

x = np.linspace(0, 2*math.pi, 1000)          # Create sequence from 0 to 2*pi
plt.plot(x, np.sin(x), "g-", x, np.cos(x), "m-")
                                             # Draw both lines in one go

## [<matplotlib.lines.Line2D object at 0x00000190AD0F8898>, <matplotlib.lines.Line2D object at

plt.legend(labels=["sin(x)", "cos(x)"])      # Add a legend

## <matplotlib.legend.Legend object at 0x00000190D82C6128>

plt.savefig("trigonometric_functions.pdf")   # Save plot as PDF file
```

https://matplotlib.org/api/_as_gen/matplotlib.pyplot.savefig.html

The API documentation of `savefig` contains the documentation of all the optional arguments of `savefig`.

seaborn

Creating plots with matplotlib's pyplot interface can sometimes be cumbersome and clunky. seaborn provides a simpler high-level interface for visualising data. Under the bonnet, seaborn uses matplotlib.

Scatter plots and line plots involving two numerical variables

The function relplot creates a scatter plot or a line plot of two variables. Key arguments of the function relplot are given in the table below.

Argument	Description
x	Numerical variable to be used for the horizontal axis
y	Numerical variable to be used for the vertical axis
hue	Numerical or categorical variable to be used to colour the observations (optional)
size	Numerical or categorical variable to be used as size of the plotting symbol or line (optional)
style	Categorical variable to be used as plotting symbol or line (optional)
data	Name of the data frame to be used to retrieve variables (optional)
kind	"scatter" (for points, default) or "line" (for lines) (optional)

Variables can either be vectors or the name of a column in the data frame supplied in the argument data.

Just like ggplot2 in R, seaborn assumes that the data is in long, rather than wide format.

seaborn provides functions for producing the different kinds of plots without having to call relplot with the argument kind. Calling relplot with kind="scatter" is equivalent to calling the function scatterplot, whereas calling relplot with kind="line" is equivalent to calling lineplot.

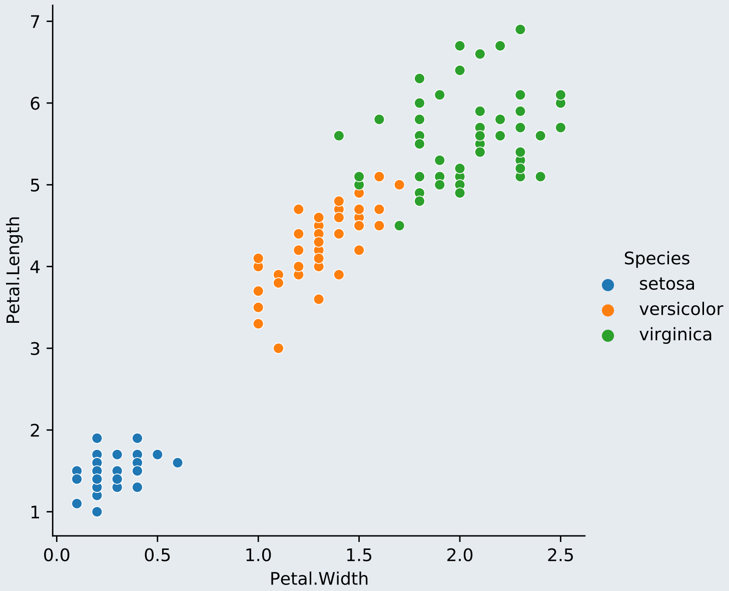
Example 16.

In this example we will plot the petal width against the petal length for the iris data.

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.relplot("Petal.Width", "Petal.Length", hue="Species", data=iris)
```

```
## <seaborn.axisgrid.FacetGrid object at 0x00000190DA9A0E80>
##
## C:\Users\Vinny\AppData\Local\Programs\Python\Python37\lib\site-packages\seaborn\_decorators.
##   FutureWarning
```

```
plt.show()
```



We could have also used the function `scatterplot`.

Example 17.

In this example we use seaborn to produce a plot showing the sine and the cosine function.

We start by creating a sequence of numbers between 0 and 2π and then add sine and cosine to obtain a wide data frame, with separate columns for sine and cosine.

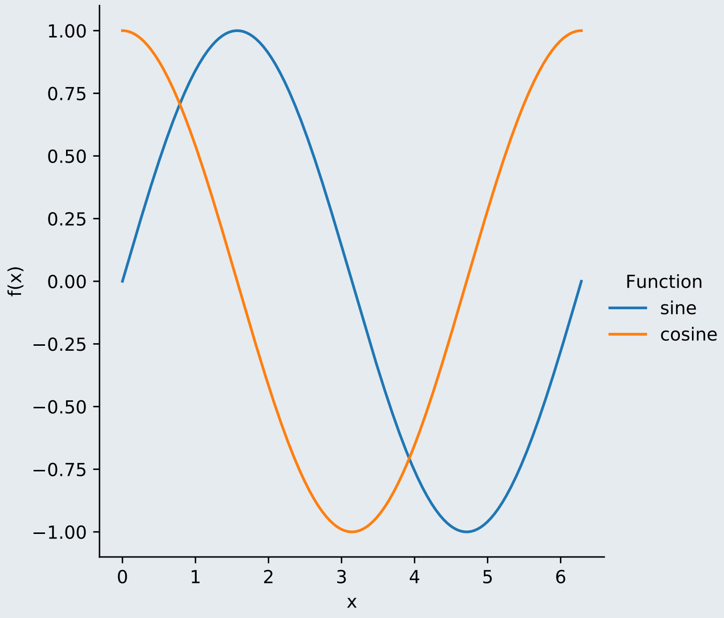
```
import numpy as np
import pandas as pd
import math
data = pd.DataFrame({"x": np.linspace(0, 2*math.pi, 1000)})
data = data.eval("sine = sin(x)")
data = data.eval("cosine = cos(x)")
```

In order to be able to plot sine and cosine in a single seaborn plot we convert the data from wide format for long format.

```
data=data.melt(id_vars=["x"], value_vars=["sine","cosine"],
               var_name="Function", value_name="f(x)")
sns.relplot("x", "f(x)", hue="Function", data=data, kind="line")
```

```
## <seaborn.axisgrid.FacetGrid object at 0x00000190DA9E60F0>
##
## C:\Users\Vinny\AppData\Local\Programs\Python\Python37\lib\site-packages\seaborn\_decorators.
##   FutureWarning
```

```
plt.show()
```



Faceting We can also use a numerical or categorical variable to split the data and show the plot for the subsets of the data in different subplots ("faceting"). The optional arguments to `relplot` to control faceting are given in the table below.

Argument	Description
`col`	Numerical or categorical variable used to subset the data across columns (optional)
`row`	Numerical or categorical variable used to subset the data across rows (optional)
`col_wrap`	If `row` is not specified, start a new row after `col_wrap` columns (optional)

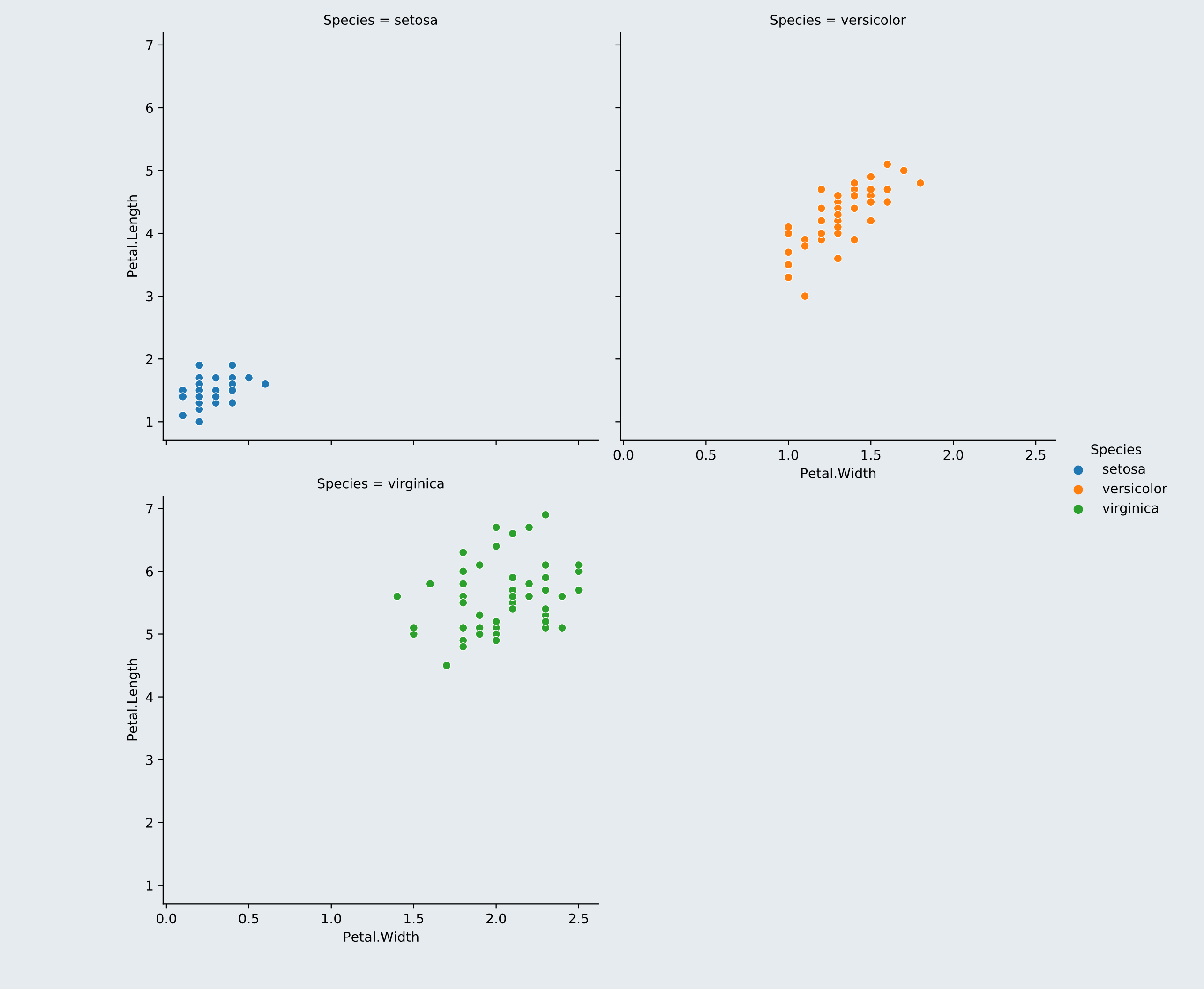
Example 18.

In this example we will plot the data for the different species in different subplots.

```
sns.relplot("Petal.Width", "Petal.Length", hue="Species", col="Species",
            col_wrap=2, data=iris)
```

```
## <seaborn.axisgrid.FacetGrid object at 0x00000190DAB98D30>
##
## C:\Users\Vinny\AppData\Local\Programs\Python\Python37\lib\site-packages\seaborn\_decorators.
##   FutureWarning
```

```
plt.show()
```



https://seaborn.pydata.org/generated/seaborn.relplot.html

The API documentation of `relplot` contains the documentation of all the optional arguments of `relplot`.

Plots involving one numerical and one categorical variable

The function `catplot` can be used to illustrate how a numerical variable depends on a categorical variable. It can produce:

- strip plots and swarm plots,
- boxplots and boxenplots, and
- violin plots.

The most important arguments of `catplot` are given the in the table below.

Argument	Description
`x`	Numerical or categorical* variable to be used for the horizontal axis
`y`	Numerical or categorical* variable to be used for the vertical axis
`hue`	Numerical or categorical variable to be used to colour the observations (optional)
`kind`	`"strip"` (default), `"swarm"`, `"box"`, `"boxen"` or `"violin"` (optional)

*One out of `x` and `y` should be numerical, one should be categorical.

`catplot` also supports faceting in the same way as `relplot`.

Rather than specifying the `kind`, one can also use the dedicated plotting functions

kind	Equivalent function name
"strip"	stripplot
"swarm"	swarmplot
"box"	boxplot
"boxen"	boxenplot
"violin"	violinplot

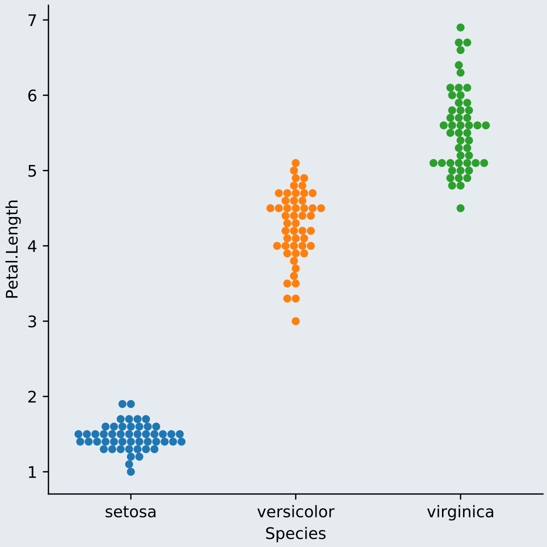
Example 19.

We will generate a swarm plot and a boxen plot for the petal length of the iris data set for the three species of iris.

```
sns.catplot("Species", "Petal.Length", hue="Species",
            data=iris, kind="swarm")
```

```
## <seaborn.axisgrid.FacetGrid object at 0x00000190DB1DA240>
##
## C:\Users\Vinny\AppData\Local\Programs\Python\Python37\lib\site-packages\seaborn\_decorators.
##   FutureWarning
```

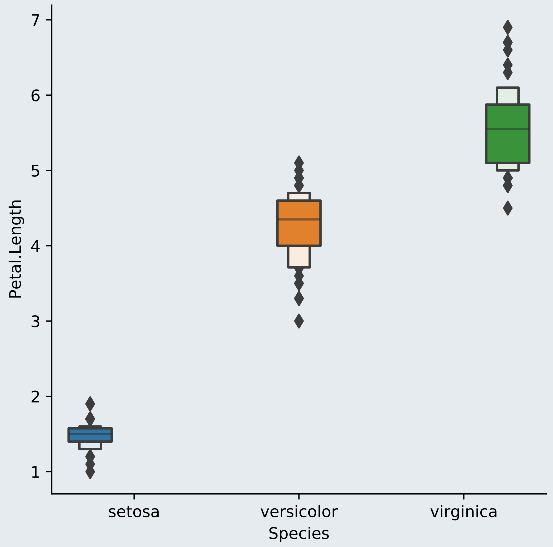
```
plt.show()
```



```
sns.catplot("Species", "Petal.Length", hue="Species",
            data=iris, kind="boxen")
```

```
## <seaborn.axisgrid.FacetGrid object at 0x00000190D4D38B38>
##
## C:\Users\Vinny\AppData\Local\Programs\Python\Python37\lib\site-packages\seaborn\_decorators.
##   FutureWarning
```

```
plt.show()
```



https://seaborn.pydata.org/generated/seaborn.catplot.html

The API documentation of `catplot` contains the documentation of all the optional arguments of `catplot`.

Plotting distributions

Univariate distributions The function `distplot` shows a histogram and/or kernel density estimate of the univariate distribution of a column in a data frame.

Optional Boolean arguments control what to display:

- a histogram (`hist=True`, default),
- a kernel density estimate (`kde=True`, default), and/or
- a rug (`rug=True`).

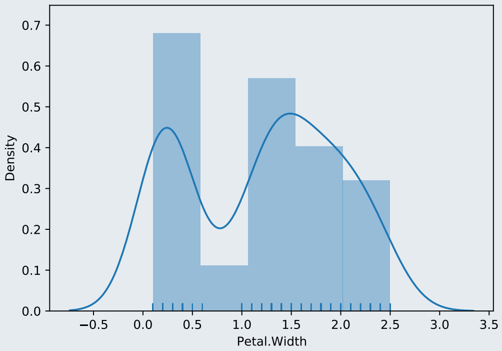
Example 20.

We can illustrate the distribution of the petal width in the iris data set using

```
sns.distplot(iris["Petal.Width"], rug=True)
```

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x00000190D7724D68>
##
## C:\Users\Vinny\AppData\Local\Programs\Python\Python37\lib\site-packages\seaborn\distribution
##   warnings.warn(msg, FutureWarning)
## C:\Users\Vinny\AppData\Local\Programs\Python\Python37\lib\site-packages\seaborn\distribution
##   warnings.warn(msg, FutureWarning)
```

```
plt.show()
```

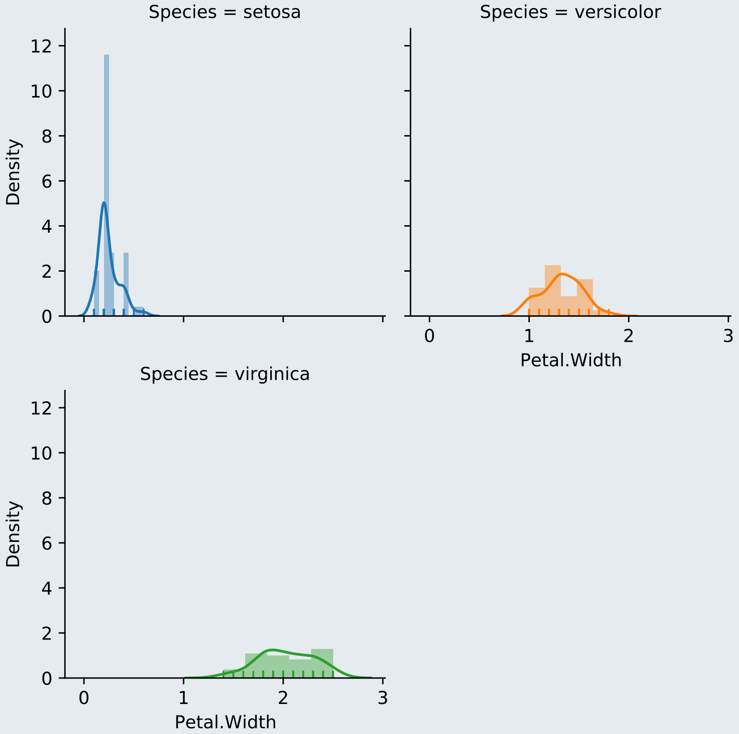


It is a little more difficult to produce separate plots for the different species. `distplot` does not (yet) have arguments `row` and `col` like say `relplot`. Seaborn however has a generic interface for faceting, the use of which is illustrated by the example below.

```
grid = sns.FacetGrid(iris, col="Species", col_wrap=2, hue="Species")
grid.map(sns.distplot, "Petal.Width", rug=True)
```

```
## <seaborn.axisgrid.FacetGrid object at 0x00000190DB1DA1D0>
##
## C:\Users\Vinny\AppData\Local\Programs\Python\Python37\lib\site-packages\seaborn\distribution
##   warnings.warn(msg, FutureWarning)
## C:\Users\Vinny\AppData\Local\Programs\Python\Python37\lib\site-packages\seaborn\distribution
##   warnings.warn(msg, FutureWarning)
## C:\Users\Vinny\AppData\Local\Programs\Python\Python37\lib\site-packages\seaborn\distribution
##   warnings.warn(msg, FutureWarning)
## C:\Users\Vinny\AppData\Local\Programs\Python\Python37\lib\site-packages\seaborn\distribution
##   warnings.warn(msg, FutureWarning)
## C:\Users\Vinny\AppData\Local\Programs\Python\Python37\lib\site-packages\seaborn\distribution
##   warnings.warn(msg, FutureWarning)
## C:\Users\Vinny\AppData\Local\Programs\Python\Python37\lib\site-packages\seaborn\distribution
##   warnings.warn(msg, FutureWarning)
```

```
plt.show()
```



https://seaborn.pydata.org/generated/seaborn.distplot.html

The API documentation of `distplot` contains the documentation of all the optional arguments of `distplot`.

https://seaborn.pydata.org/generated/seaborn.FacetGrid.html

The API documentation of `FacetGrid` explains in more detail how to set up custom faceting.

Bivariate distributions The function `jointplot` produces plots illustrating the joint distribution of two columns together with plots showing the marginal distribution on the sides.

The most important arguments to `jointplot` are given in the table below.

Argument	Description
`x`	Numerical variable to be used for the horizontal axis
`y`	Numerical variable to be used for the vertical axis
`data`	Name of the data frame to be used to retrieve variables
`kind`	`"scatter"` (for points, default) or `"reg"` (for regression), `"resid"` (for residuals), `"kde"` (for kernel density estimate), `"hex"` (for hexagonal bin plot)

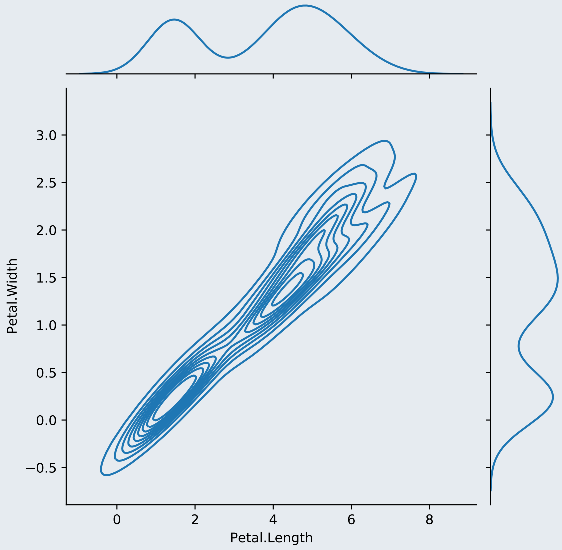
Example 21.

We can visualise the joint distribution of the petal length and the petal width in the iris data set using the following code.

```
sns.jointplot("Petal.Length", "Petal.Width", data=iris, kind="kde")
```

```
## <seaborn.axisgrid.JointGrid object at 0x00000190D16F0B38>
##
## C:\Users\Vinny\AppData\Local\Programs\Python\Python37\lib\site-packages\seaborn\_decorators.
##   FutureWarning
```

```
plt.show()
```



https://seaborn.pydata.org/generated/seaborn.jointplot.html

The API documentation of `jointplot` contains the documentation of all the optional arguments of `jointplot`.

Pair plots

The function `pairplot` produces a pair plot, i.e. a matrix of scatter plots of the numerical columns of a data frame. It takes as first argument the data frame to be plotted. The optional argument `hue` can use used to plot observations in different colours.

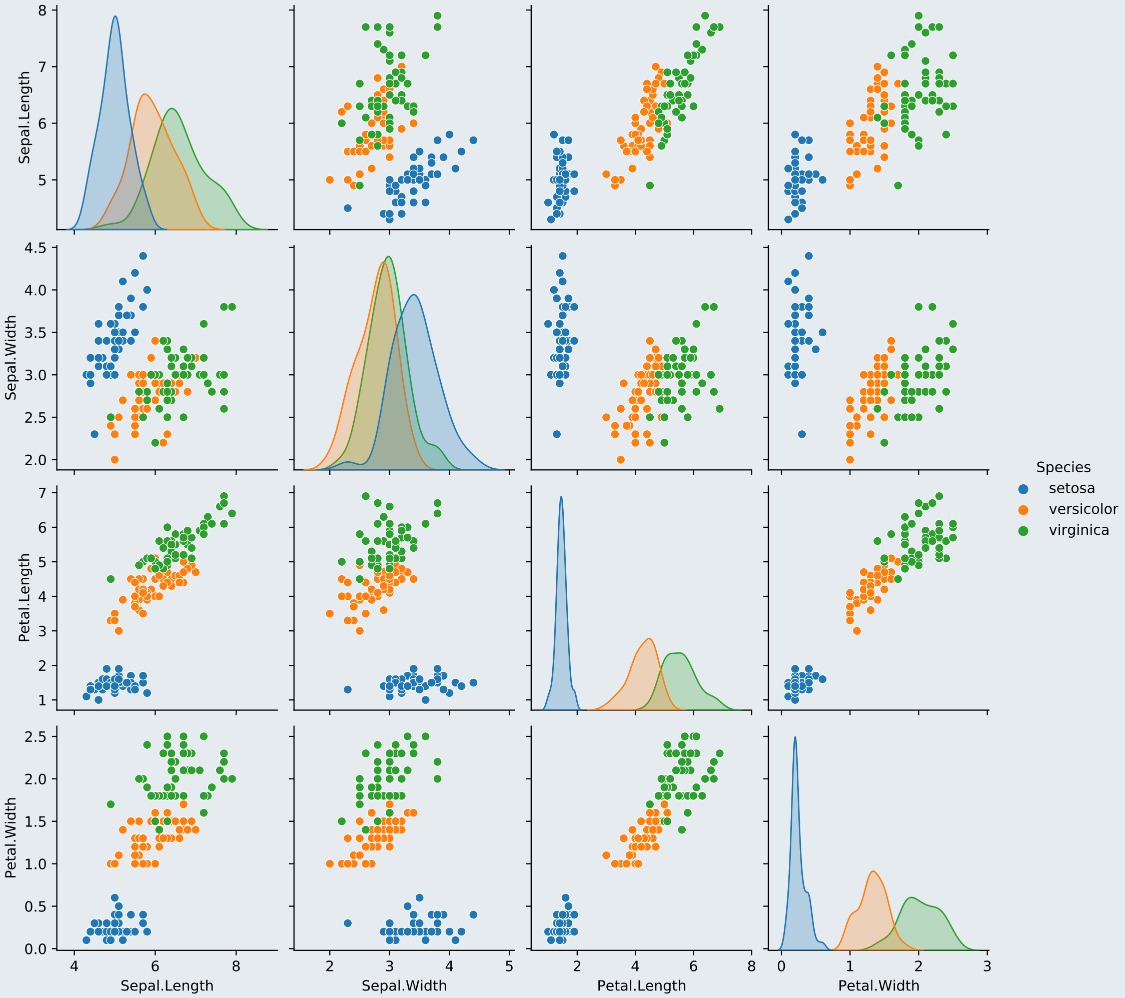
Example 22.

We can produce a scatter plots of all numerical variables in the iris data set using the following code.

```
sns.pairplot(iris, hue="Species")
```

```
## <seaborn.axisgrid.PairGrid object at 0x00000190D82A2DA0>
```

```
plt.show()
```



https://seaborn.pydata.org/generated/seaborn.pairplot.html

The API documentation of `pairplot` contains the documentation of all the optional arguments of `pairplot`.

More flexible grid layouts

https://seaborn.pydata.org/generated/seaborn.PairGrid.html

seaborn allows for much more flexible grid layouts using the `PairGrid` function.

Task 4.

Seaborn has a built-in data set on the amount of tips given by diners in the US.

```
tips = sns.load_dataset("tips")
print(tips.head())
```

```
##    total_bill   tip     sex smoker  day    time  size
## 0       16.99  1.01  Female     No  Sun  Dinner     2
## 1       10.34  1.66    Male     No  Sun  Dinner     3
## 2       21.01  3.50    Male     No  Sun  Dinner     3
## 3       23.68  3.31    Male     No  Sun  Dinner     2
## 4       24.59  3.61  Female     No  Sun  Dinner     4
```

- Create a scatter plot of the tip against the total bill, using colour to distinguish between lunch and dinner.
- Create a new variable `tip_percent` in the data frame, which is $\frac{\texttt{tip}}{\texttt{total\_bill}} \times 100$.
- Create a box plot or violin plot of the variable you have just created for lunch and dinner.

Pandas

pandas has some limited built-in plotting functions, which can provide a quick way of visualising the information in a data frame.

In terms of functionality, the functions and methods are however closer to matplotlib's pyplot than seaborn. Most importantly, pandas's plotting functions cannot use a grouping variable in the same way as seaborn can.

Plotting data from a series

Series have a limited built-in plotting interface.

The method plot can draw different types plots using the data in the series.

The argument kind controls the type of plot to be generated.

Kind	Plot produced
kind="area"	Area plot using the values as y coordinates
kind="bar"	(Vertical) bar chart using the values as y coordinates
kind="barh"	Horizontal bar chart using the values as x coordinates
kind="box"	Box plot of the values in the series
kind="density" or kind="kde"	Estimated density of the values in the series
kind="hist"	Histogram of the values in the series
kind="line"	Line chart using the values as y coordinates
kind="pie"	Pie chart using the values as (unnormalised) proportions

Instead of specifying the kind as an argument to plot we can also call an equivalent method, which has the kind as its name. For example, instead of using s.plot(kind="box") we can also use s.plot.box().

https://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.plot.html

The API documentation of the plot method contains the documentation of all the optional arguments.

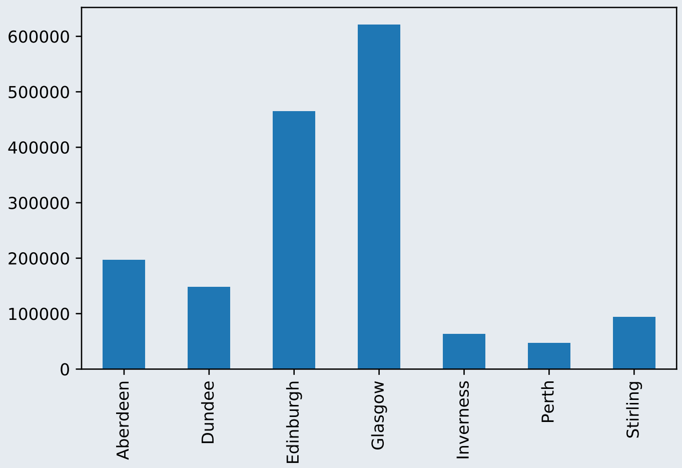
Example 23.

We can draw a bar plot illustrating the population in Scotland's seven cities using the Python code below.

```
import matplotlib.pyplot as plt
import pandas as pd

population = pd.Series( [ 196670, 148270, 464990, 621020,
                                        63780, 47180, 94000 ] ,
                         index = ["Aberdeen", "Dundee", "Edinburgh", "Glasgow",
                                  "Inverness", "Perth", "Stirling"] )
population.plot.bar()                      # Create bar plot of values
# population.plot(kind="bar")              # Equivalent alternative
```

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x00000190D7C7F7B8>
```

```
plt.subplots_adjust(bottom=0.2)            # Increase margin at the bottom
plt.show()                                 # Show plot
```



Plotting data from data frames

Pandas data frames also have a `plot` method. Its most important arguments are summarised in the table below.

Argument	Description
`x`	Index (label) of horizontal coordinate
`y`	Index (label) of vertical coordinate (can also be a list)
`kind`	see below
`subplots`	Create separate plots for the columns (default `False`)

Depending on the `kind` of the plot `x` and/or `y` have to be specified. For some kinds, omitting `x` and `y` will yield a plot of all columns.

The argument `kind` controls the type of plot to be generated.

Kind	Plot produced
`kind="line"`	Line plot through the coordinates given by the columns
`kind="scatter"`	Scatter plot using the coordinates given by the columns
`kind="area`	Stacked area plot
`kind="bar"`	(Vertical) bar chart using the values as y coordinates
`kind="barh"`	Horizontal bar chart using the values as x coordinates
`kind="box"`	Box plot of the selected columns
`kind="density"` or `kind="kde"`	Estimated density of the values in the series
`kind="hexbin"`	Hex bin plot of of the values in the columns
`kind="hist"`	Histogram of the values in the columns
`kind="pie"`	Pie chart using the values as (unnormalised) proportions

Just like for series we can use the alternative syntax `d.plot.box()` instead of `d.plot(kind="box")`.

https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.plot.html

The API documentation of the `plot` method contains the documentation of all the optional arguments.

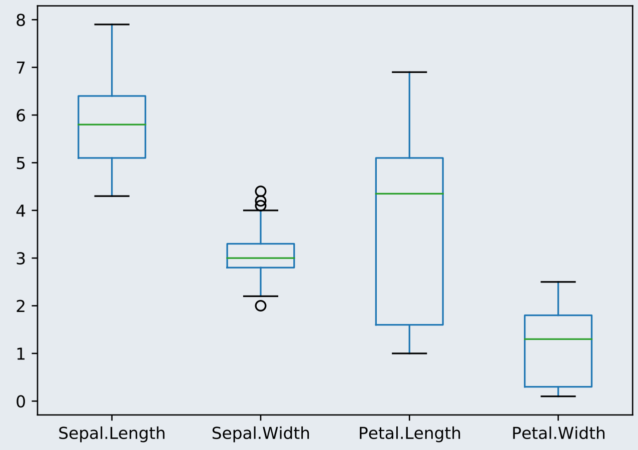
Example 24.

We can create box plots for the distribution of all four numerical columns (species confounded) using

```
iris.plot.box()
```

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x00000190D7FFCA20>
```

```
plt.show()
```

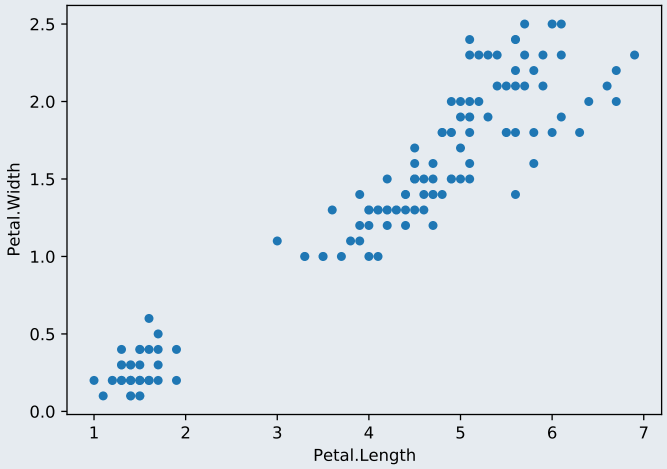


We create a scatter plot of the petal width against the petal length (all species confounded) using

```
iris.plot.scatter("Petal.Length", "Petal.Width")
```

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x00000190DABF1F60>
```

```
plt.show()
```

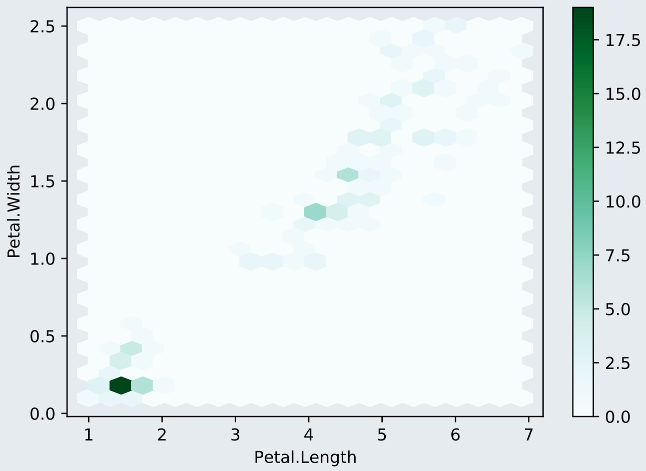


We can produce a hexbin plot using

```
iris.plot.hexbin("Petal.Length", "Petal.Width", gridsize=(20,15))
```

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x00000190DAD5C4E0>
```

```
plt.show()
```



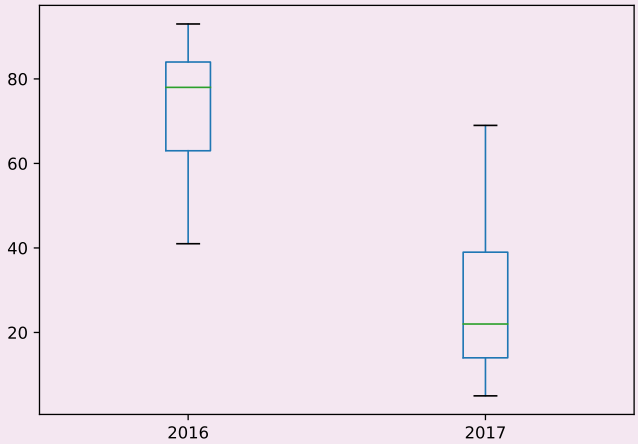
It is difficult to plot observations of the different species in different colours. It is much easier to use `seaborn` in that case.

Task 5.

The file `TRUMPWORLD-pres.csv` (available from https://raw.githubusercontent.com/fivethirtyeight/data/master/trump-world-trust/TRUMPWORLD-pres.csv) contains the percentage of the population of different countries who have a positive view of the US president. It is the data behind the story "What the world thinks of Donald Trump on fivethirtyeight.com.

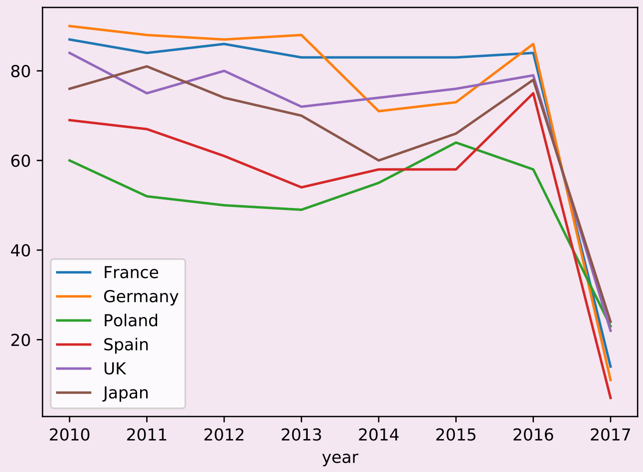
- Read the data into Python.
- Remove all data from before the year 2010 and remove the column `avg`.
- Make the column `year` the row index of the data frame.
- Create box plots of the ratings for the years 2016 and 2017. Your plot should look similar to the one shown below. Note that pandas plots are based on the columns of a data set and not the rows, so you have to transpose the data frame before creating the box plots.

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x00000190D0E92C50>
```



- Remove all columns which have missing values.
- Create a line plot of the time series for the UK.
- Create a line plot of the time series for all countries with complete data using different lines and colours for the different countries. The plot should look similar to the one shown below.

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x00000190DAB71978>
```



Other plots

Pair plots The function `plotting.scatter_matrix(d)` can be used to produce a scatter plot matrix ("pair plots") of the columns of `d`.

https://pandas.pydata.org/pandas-docs/stable/generated/pandas.plotting.scatter_matrix.html

The API documentation of the function `scatter_matrix` method contains the documentation of all the optional arguments.

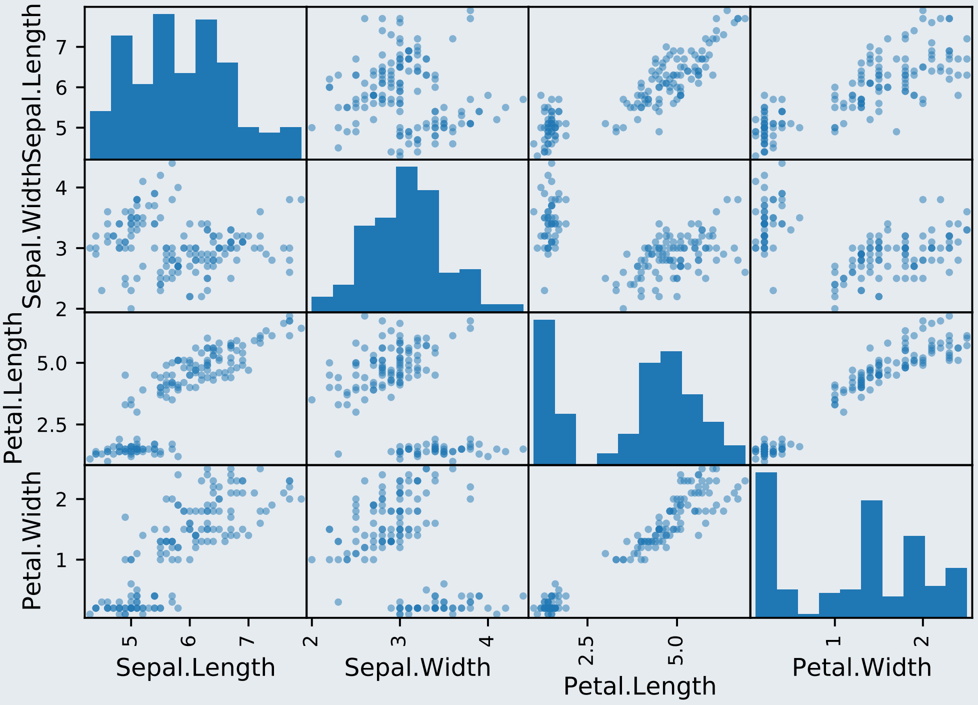
Example 25.

We can produce a scatter plot matrix of the iris data using

```
pd.plotting.scatter_matrix(iris)
```

```
## array([[<matplotlib.axes._subplots.AxesSubplot object at 0x00000190DAD6CB00>,
##         <matplotlib.axes._subplots.AxesSubplot object at 0x00000190D7F70A20>,
##         <matplotlib.axes._subplots.AxesSubplot object at 0x00000190D0F35E48>,
##         <matplotlib.axes._subplots.AxesSubplot object at 0x00000190DB0D0208>],
##        [<matplotlib.axes._subplots.AxesSubplot object at 0x00000190DB0FF588>,
##         <matplotlib.axes._subplots.AxesSubplot object at 0x00000190DB12F908>,
##         <matplotlib.axes._subplots.AxesSubplot object at 0x00000190DB15FC88>,
##         <matplotlib.axes._subplots.AxesSubplot object at 0x00000190DB5A1FD0>],
##        [<matplotlib.axes._subplots.AxesSubplot object at 0x00000190DB5B0080>,
##         <matplotlib.axes._subplots.AxesSubplot object at 0x00000190DB610748>,
##         <matplotlib.axes._subplots.AxesSubplot object at 0x00000190DB640AC8>,
##         <matplotlib.axes._subplots.AxesSubplot object at 0x00000190DC154E48>],
##        [<matplotlib.axes._subplots.AxesSubplot object at 0x00000190DC191208>,
##         <matplotlib.axes._subplots.AxesSubplot object at 0x00000190DC1C3588>,
##         <matplotlib.axes._subplots.AxesSubplot object at 0x00000190DC1F3908>,
##         <matplotlib.axes._subplots.AxesSubplot object at 0x00000190DC226C88>]],
##       dtype=object)
```

```
plt.show()
```



Parallel coordinate plots The function `parallel_coordinates(d, class_column)` can be used to produce a parallel coordinates plot of the columns of `d` using `class_column` to draw the lines in different colours.

A parallel coordinate plot is rather unusual in the sense that each observation corresponds to a line.

https://pandas.pydata.org/pandas-docs/stable/generated/pandas.plotting.parallel_coordinates.html

The API documentation of the function `parallel_coordinates` method contains the documentation of all the optional arguments.

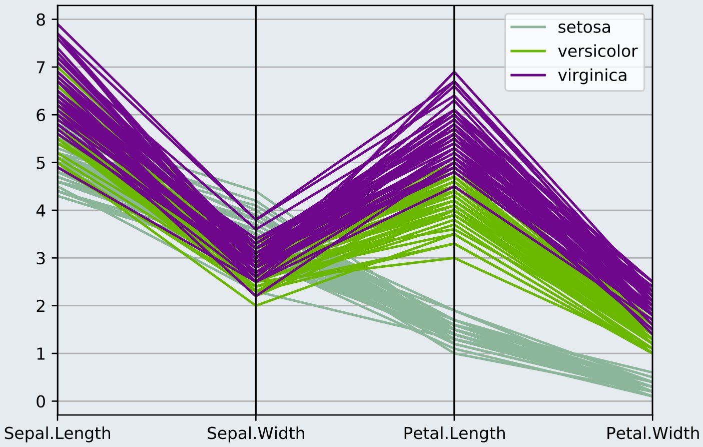
Example 26.

We can produce a parallel coordinate plot of the iris data using the code below.

```
pd.plotting.parallel_coordinates(iris, "Species")
```

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x00000190D7C184A8>
```

```
plt.show()
```


8
7
6
5
4
3
2
1
0
Sepal.Length
Sepal.Width
Petal.Length
Petal.Width
setosa
versicolor
virginica

Review tasks

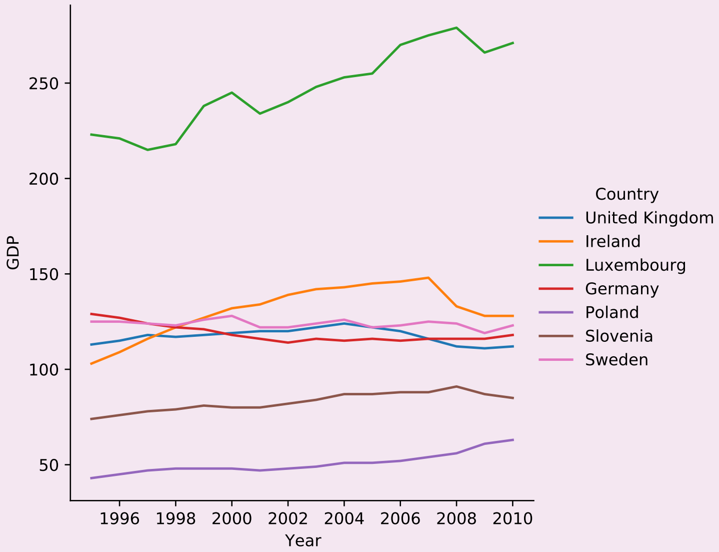
Task 6.

The data file `eu2.csv` (available from https://github.com/UofGAnalyticsData/DPIP) contains the per-capita gross domestic product (GDP) in PPS of seven EU countries. PPS (purchasing power standard) is an artificial currency unit used by Eurostat. One PPS can buy the same amount of goods and services in each country.

Create a line plot of the PPS in the 6 countries.

Your plot should look similar to the one shown below.

```
## <seaborn.axisgrid.FacetGrid object at 0x00000190D3C13DA0>
##
## C:\Users\Vinny\AppData\Local\Programs\Python\Python37\lib\site-packages\seaborn\_decorators.
##   FutureWarning
```



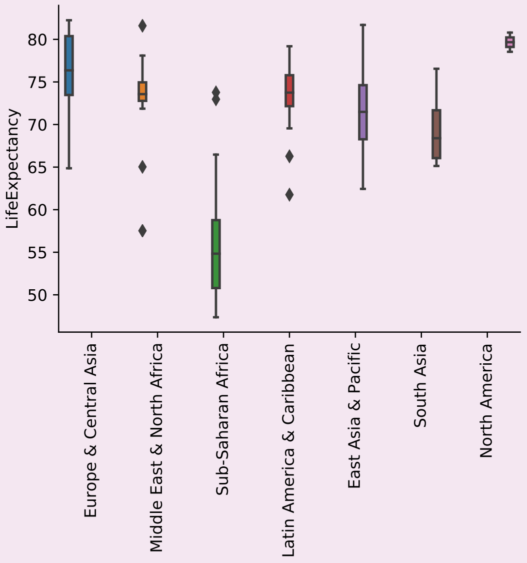
Task 7.

The data file `health.csv` (available from https://github.com/UofGAnalyticsData/DPIP) contains data on health expenditure and life expectancy for large selection of countries.

Create box plots of the life expectancy for the different regions. Your plot should look like the plot shown below.

```
## <seaborn.axisgrid.FacetGrid object at 0x00000190DAA56C88>
##
## C:\Users\Vinny\AppData\Local\Programs\Python\Python37\lib\site-packages\seaborn\_decorators.
##   FutureWarning
```

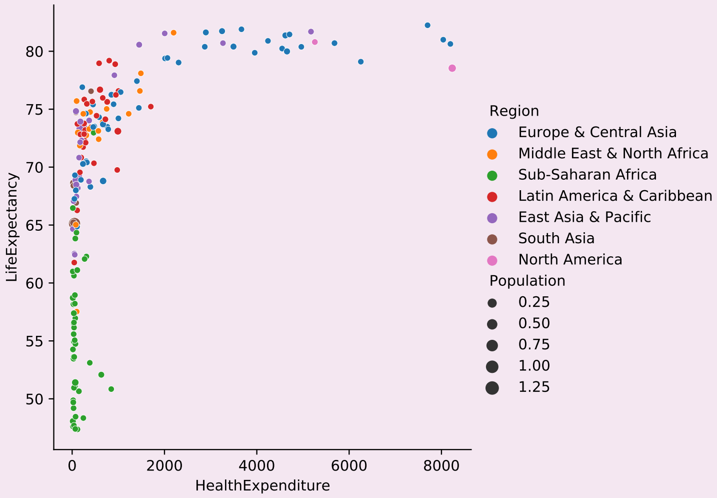
```
## (array([0, 1, 2, 3, 4, 5, 6]), <a list of 7 Text major ticklabel objects>)
```



Create a scatter plot of the life expectancy against the health expenditure using different colour for different regions. The size of the marker should reflect the population size of the country.

Your plot should look like the plot shown below.

```
## <seaborn.axisgrid.FacetGrid object at 0x00000190D774B438>
##
## C:\Users\Vinny\AppData\Local\Programs\Python\Python37\lib\site-packages\seaborn\_decorators.
##   FutureWarning
```



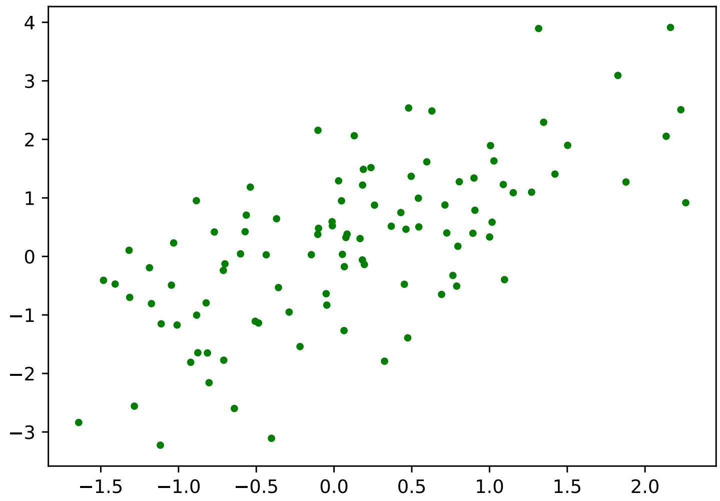
Answers to tasks

Answer to Task 1. We can use the following Python code.

```
x = np.random.standard_normal(100)      # Create a random data cloud
y = x + np.random.standard_normal(100)
plt.plot(x, y, marker=".", color="g", linestyle="")
                                        # Draw scatter plot
```

```
## [<matplotlib.lines.Line2D object at 0x00000190DAA91780>]
```

```
plt.show()                              # Show plot
```

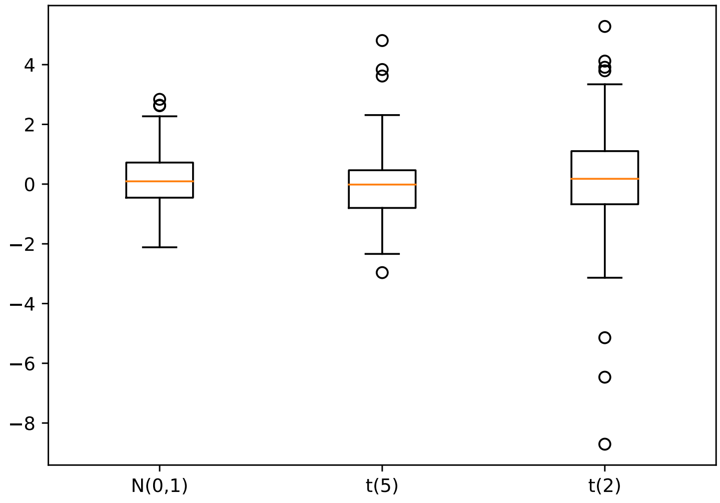


Answer to Task 2. We can use the following Python code.

```
x1 = np.random.standard_normal(100)
x2 = np.random.standard_t(5, 100)
x3 = np.random.standard_t(2, 100)
plt.boxplot([x1, x2, x3],                        # Create box plots
            labels = ["N(0,1)", "t(5)", "t(2)"])
```

```
## {'whiskers': [<matplotlib.lines.Line2D object at 0x00000190DADAE860>, <matplotlib.lines.Line2D objec
```

```
plt.show()                                       # Show plot
```



We can see that the lower the degrees of the freedom of the t-distribution, the heavier the tails will be (i.e. the more outliers will show up in the box plot).

Answer to Task 3. We can use the following Python code.

```
fig, ax = plt.subplots(2, 2, sharex=True, sharey=True)
                                        # Split the plotting region and share y axis
ax[0][0].plot("Petal.Length", "Petal.Width", ".", color="blue",
               data=iris.query("Species=='setosa'"))
```

```
## [<matplotlib.lines.Line2D object at 0x00000190D28B2668>]
```

```
ax[0][0].set_title("Setosa")
```

```
## Text(0.5, 1.0, 'Setosa')
```

```
ax[0][1].plot("Petal.Length", "Petal.Width", ".", color="purple",
               data=iris.query("Species=='versicolor'"))
```

```
## [<matplotlib.lines.Line2D object at 0x00000190D7774FD0>]
```

```
ax[0][1].set_title("Versicolor")
```

```
## Text(0.5, 1.0, 'Versicolor')
```

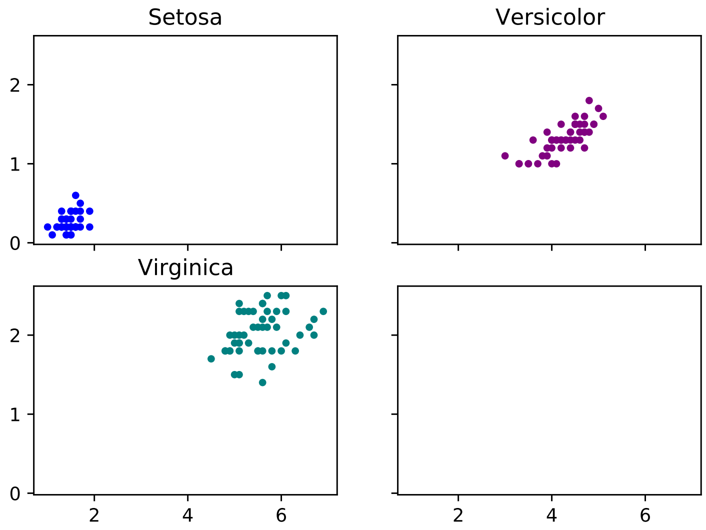
```
ax[1][0].plot("Petal.Length", "Petal.Width", ".", color="teal",
               data=iris.query("Species=='virginica'"))
```

```
## [<matplotlib.lines.Line2D object at 0x00000190D7816710>]
```

```
ax[1][0].set_title("Virginica")
```

```
## Text(0.5, 1.0, 'Virginica')
```

```
plt.show()
```

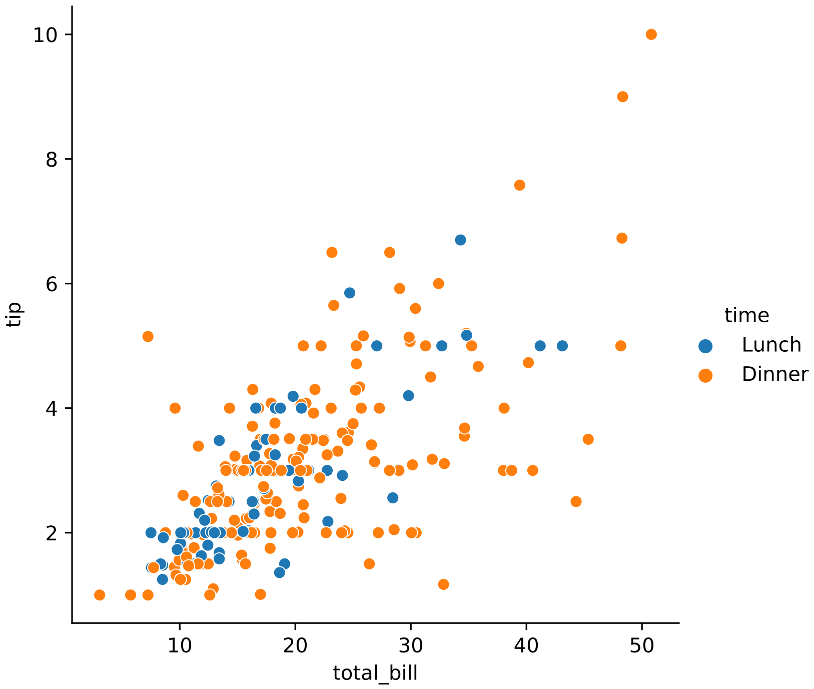


Answer to Task 4. We can use the following Python code for the first plot.

```
tips = sns.load_dataset("tips")
sns.relplot("total_bill", "tip", hue="time", data=tips)
```

```
## <seaborn.axisgrid.FacetGrid object at 0x00000190DADE7DA0>
##
## C:\Users\Vinny\AppData\Local\Programs\Python\Python37\lib\site-packages\seaborn\_decorators.py:43: F
##   FutureWarning
```

```
plt.show()
```



We next add the new column

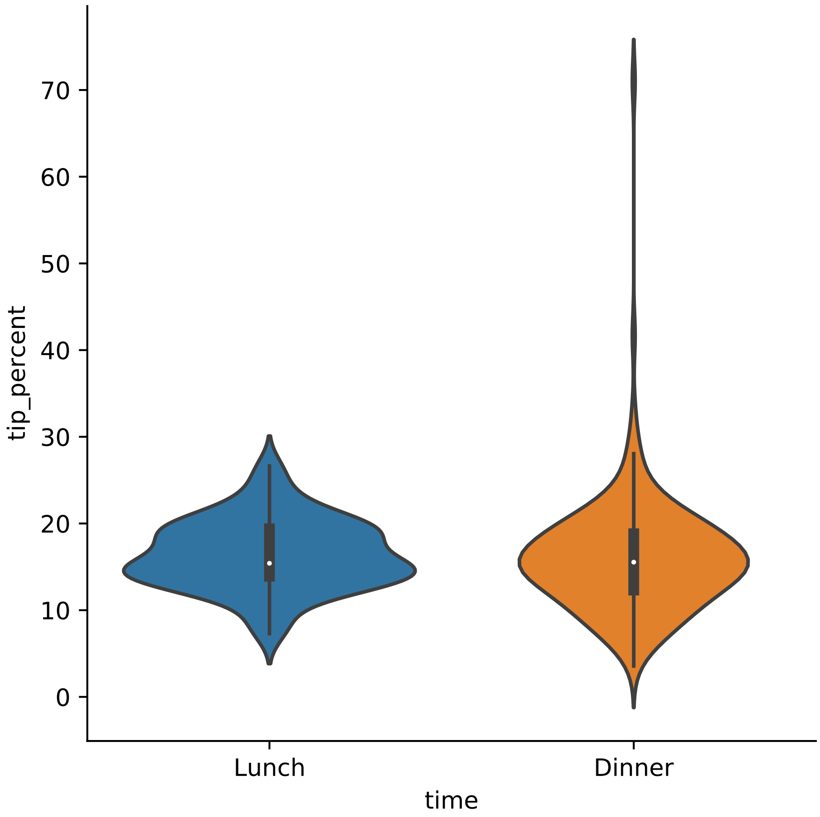
```
tips = tips.eval("tip_percent = tip / total_bill * 100")
```

We can then create the violin plots.

```
sns.catplot("time", "tip_percent", data=tips, kind="violin")
```

```
## <seaborn.axisgrid.FacetGrid object at 0x00000190D7CA7860>
##
## C:\Users\Vinny\AppData\Local\Programs\Python\Python37\lib\site-packages\seaborn\_decorators.py:43: F
##   FutureWarning
```

```
plt.show()
```

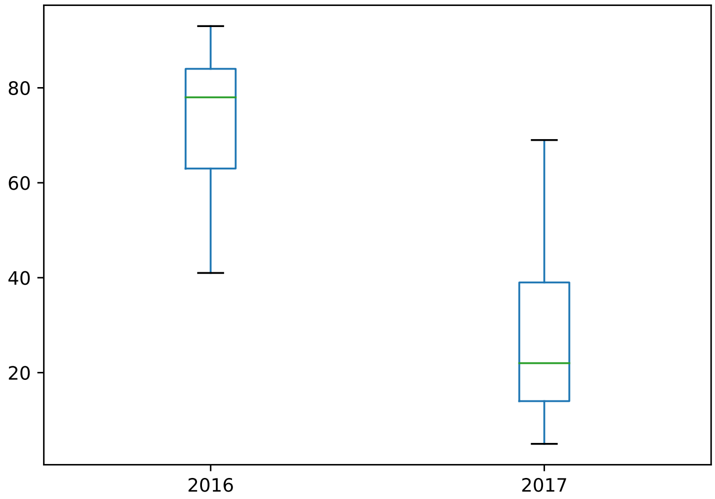


Answer to Task 5. We will start with the data management.

```
potus = pd.read_csv("TRUMPWORLD-pres.csv")
potus = potus.query("year>=2010")
potus = potus.set_index("year")
del potus["avg"]
```

For the box plots we first only select the rows corresponding to the years 2016 and 2017 and then transpose the data (so that rows become columns and vice versa).

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x00000190D7B9DCC0>
```



For the line plots we remove the columns with missing values.

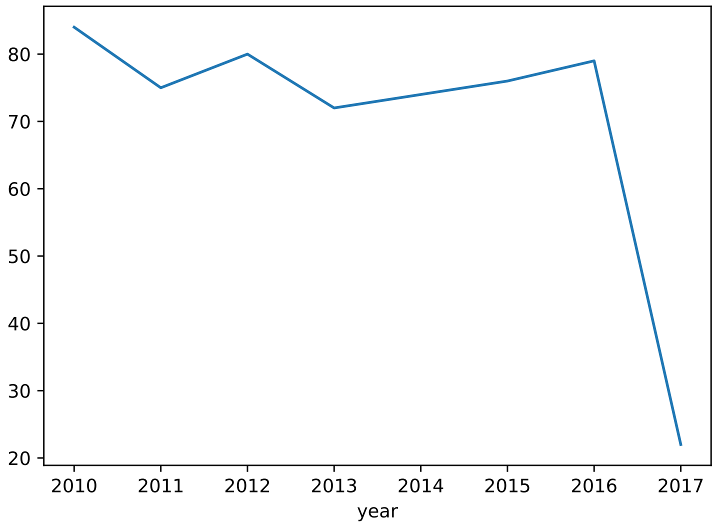
```
potus = potus.dropna(axis=1)
```

We can create the line plot for the UK using

```
potus["UK"].plot.line()
```

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x00000190D7B926A0>
```

```
plt.show()
```

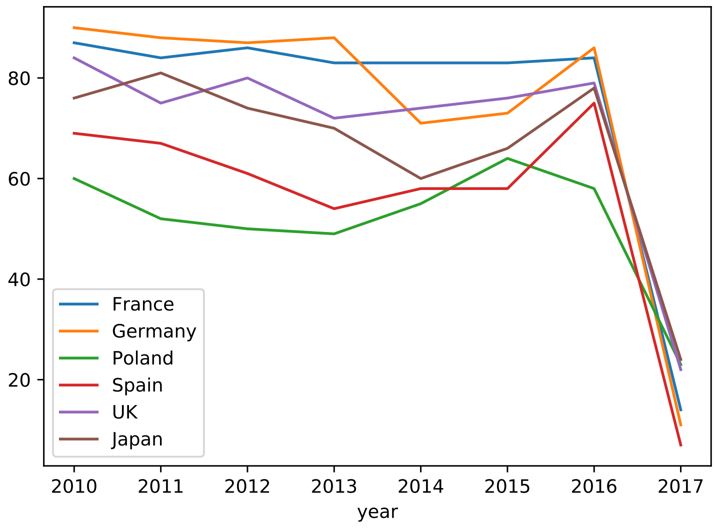


We can create the plot for all countries using

```
potus.plot.line()
```

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x00000190DB1A68D0>
```

```
plt.show()
```



Answer to Task 6. We start by reading in the data

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
eu = pd.read_csv("eu2.csv")
```

Using matplotlib (least elegant) …

```
colours = ["yellow", "orange", "red", "pink", "purple", "blue",
           "teal", "green"]
```

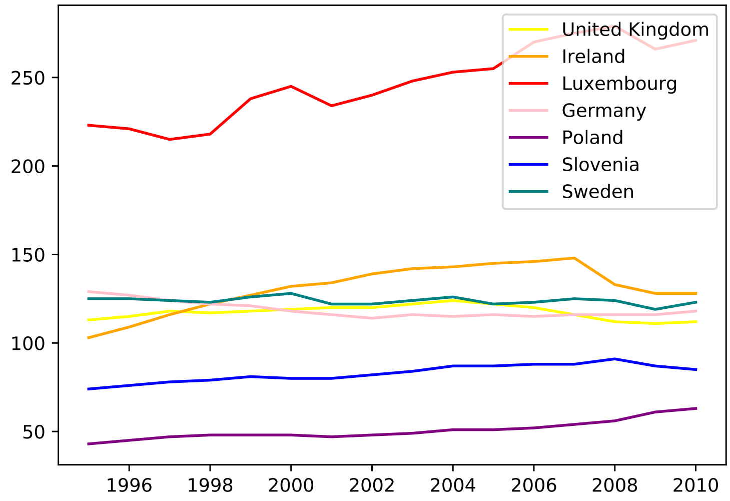
```
for i, column in enumerate(eu.columns[1:]):
    plt.plot("Year", column, data=eu, color=colours[i])
```

```
## [<matplotlib.lines.Line2D object at 0x00000190D7900978>]
## [<matplotlib.lines.Line2D object at 0x00000190D78D1CC0>]
## [<matplotlib.lines.Line2D object at 0x00000190D78D1320>]
## [<matplotlib.lines.Line2D object at 0x00000190D78D1400>]
## [<matplotlib.lines.Line2D object at 0x00000190D78D1240>]
## [<matplotlib.lines.Line2D object at 0x00000190D78D1128>]
## [<matplotlib.lines.Line2D object at 0x00000190D78B20F0>]
```

```
plt.legend(eu.columns[1:])
```

```
## <matplotlib.legend.Legend object at 0x00000190D8544AC8>
```

```
plt.show()
```

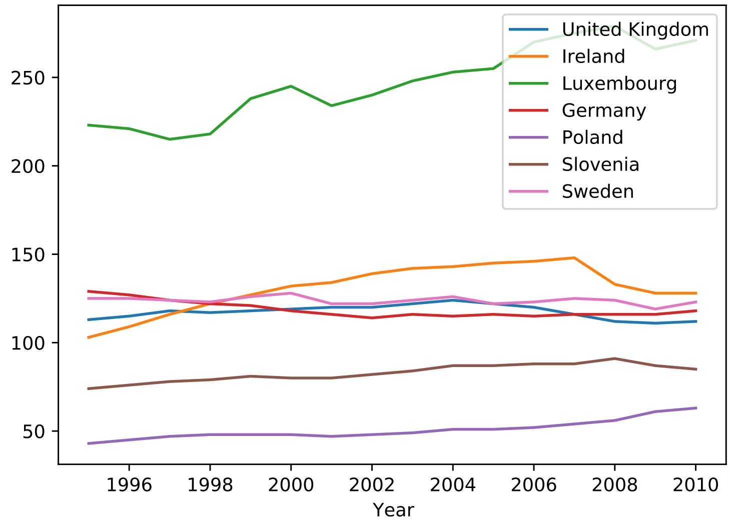


Using pandas …

```
eu.plot.line(x="Year")
```

```
## <matplotlib.axes._subplots.AxesSubplot object at 0x00000190D791AEF0>
```

```
plt.show()
```

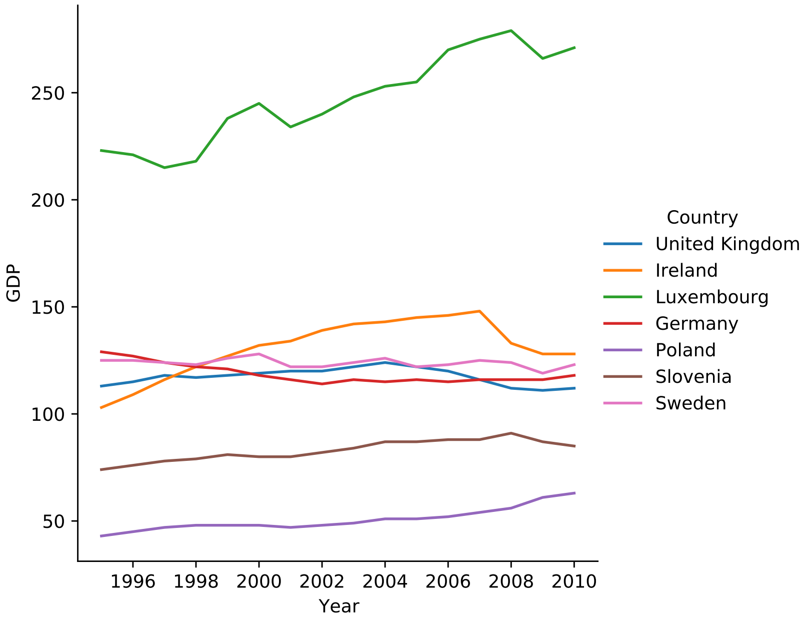


Using seaborn (which requires the data in long, rather than wide format)…

```
eu_long = eu.melt("Year", var_name="Country", value_name="GDP")
sns.relplot("Year", "GDP", kind="line", hue="Country", data=eu_long)
```

```
## <seaborn.axisgrid.FacetGrid object at 0x00000190DC24DE10>
##
## C:\Users\Vinny\AppData\Local\Programs\Python\Python37\lib\site-packages\seaborn\_decorators.py:43: F
##   FutureWarning
```

```
plt.show()
```



Answer to Task 7. We start by reading in the data.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
health = pd.read_csv("health.csv")
```

Given that the data is in long format, it is easiest to use seaborn.

We can create the first plot using …

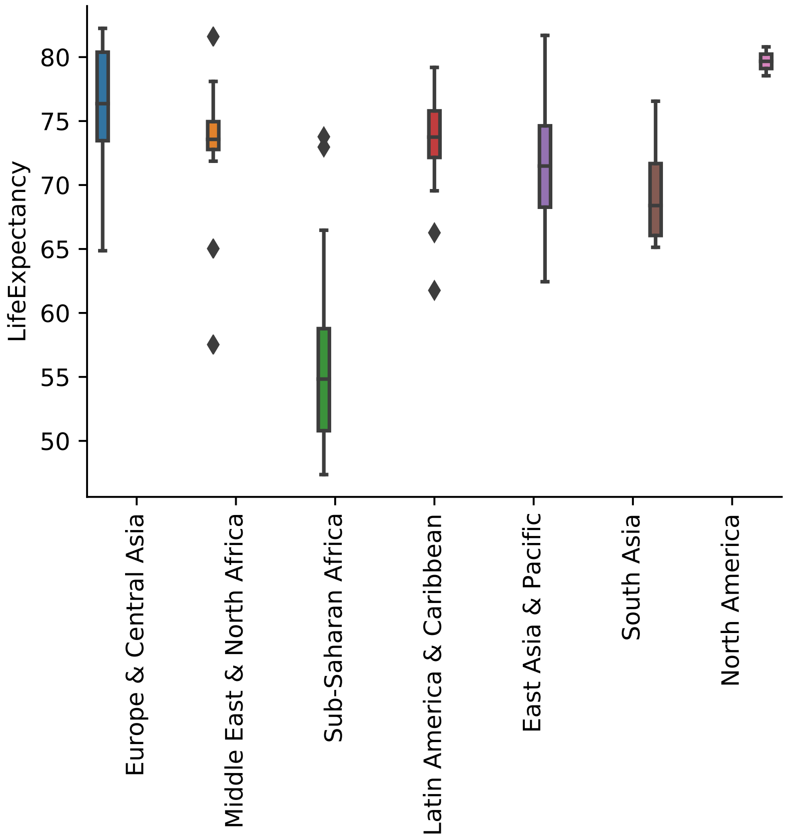
```
sns.catplot("Region", "LifeExpectancy", hue="Region", data=health, kind="box")
```

```
## <seaborn.axisgrid.FacetGrid object at 0x00000190DC1D5EF0>
##
## C:\Users\Vinny\AppData\Local\Programs\Python\Python37\lib\site-packages\seaborn\_decorators.py:43: F
##   FutureWarning
```

```
plt.xticks(rotation=90)                       # Rotate x-axis labels
```

```
## (array([0, 1, 2, 3, 4, 5, 6]), <a list of 7 Text major ticklabel objects>)
```

```
plt.subplots_adjust(bottom=0.4)               # Increase margin at the bottom
plt.show()
```

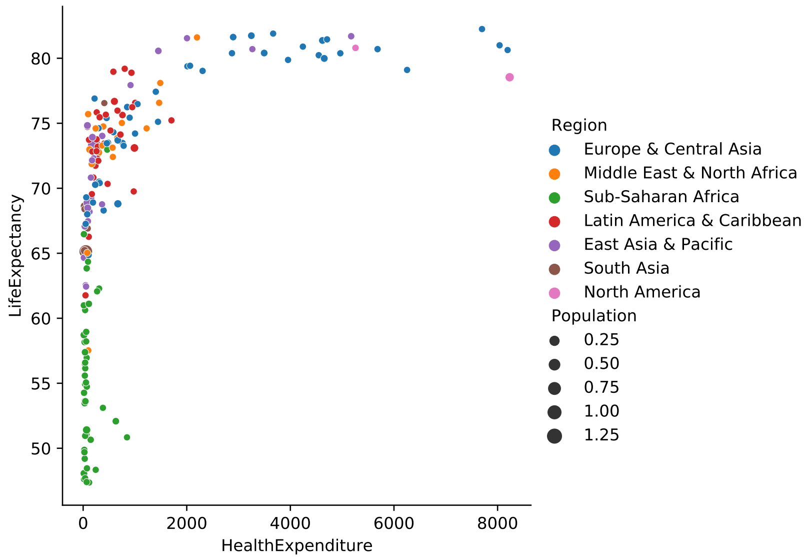


We can then create the second plot using …

```
sns.relplot("HealthExpenditure", "LifeExpectancy", hue="Region", size="Population", data=health)
```

```
## <seaborn.axisgrid.FacetGrid object at 0x00000190D8605F98>
##
## C:\Users\Vinny\AppData\Local\Programs\Python\Python37\lib\site-packages\seaborn\_decorators.py:43: F
##   FutureWarning
```

```
plt.show()
```



To create the boxplots using matplotlib, we first need to create a list containing the life expectancies for the different regions.

```
regions = health["Region"].unique()           # Get region names
regional_data = list(map(lambda region:
```

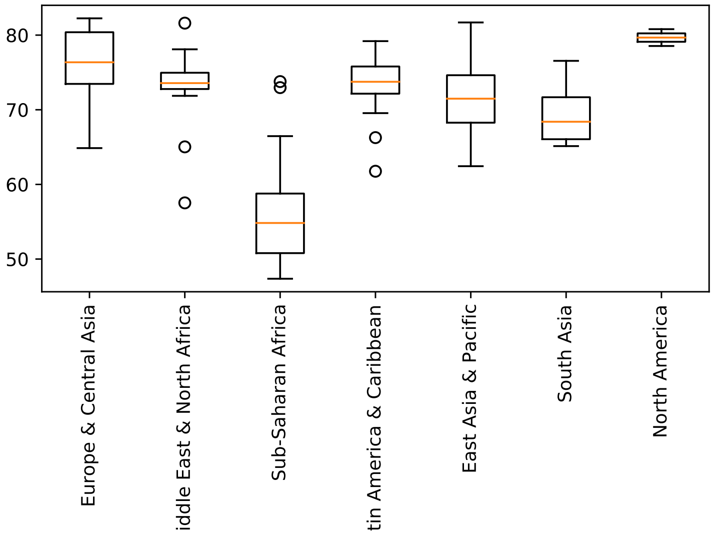
```
                                   health.query("Region=='{}'".format(region))["LifeExpectancy"],
                               regions))
plt.boxplot(regional_data, labels=regions)
```

```
## {'whiskers': [<matplotlib.lines.Line2D object at 0x00000190D0EAF198>, <matplotlib.lines.Line2D objec
```

```
plt.xticks(rotation=90)                              # Rotate x-axis labels
```

```
## (array([1, 2, 3, 4, 5, 6, 7]), <a list of 7 Text major ticklabel objects>)
```

```
plt.subplots_adjust(bottom=0.4)                      # Increase margin at the bottom
plt.show()
```

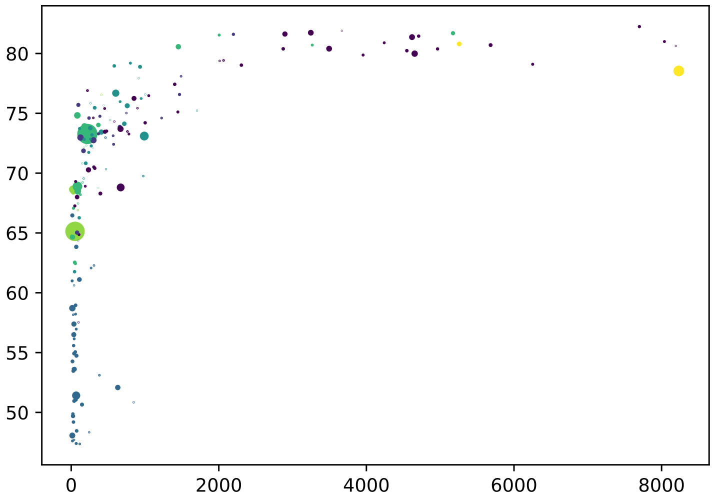


We could have created the second plot using the function `scatter` from `matplotlib`. For this, we would have to translate regions to colours though, which is quite cumbersome. We will automatically generate a sequence of colours using the `get_cmap` function. We also have to manually rescale the marker size.

```
regions =  health["Region"].unique()
colours = plt.cm.get_cmap("viridis", len(regions)).colors
colours_map = dict(zip(regions, colours))
region_colours = health["Region"].map(colours_map)
marker_size = 100 * health["Population"] / health["Population"].max()
plt.scatter("HealthExpenditure", "LifeExpectancy", color=region_colours, s=marker_size, data=health)
```

```
## <matplotlib.collections.PathCollection object at 0x00000190D78E68D0>
```

```
plt.show()
```


80
75
70
65
60
55
50
0
2000
4000
6000
8000