COMP47670

# Data Visualisation and Plotting

Slides by Derek Greene

UCD School of Computer Science

Spring 2018

## Overview

- Visualisation with Matplotlib
- Simple Matplotlib Plots
- Plotting with Pandas
- Other Visualisation Packages

## Visualisation with Matplotlib

- Data visualisation is a key part of the exploratory data analysis process - try to figure out what story the data has to tell.
- Matplotlib: The standard Python plotting library. Provides a variety of plot types. Included by default in the Anaconda distribution: http://matplotlib.org/users/beginner.html
- Matplotlib supports the customisation of plot appearance. You can control the defaults of almost every property: figure size, line width, colour and style, axes, axis and grid properties, text and font properties and so on.
- Matplotlib can draw plots directly in IPython notebooks.

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

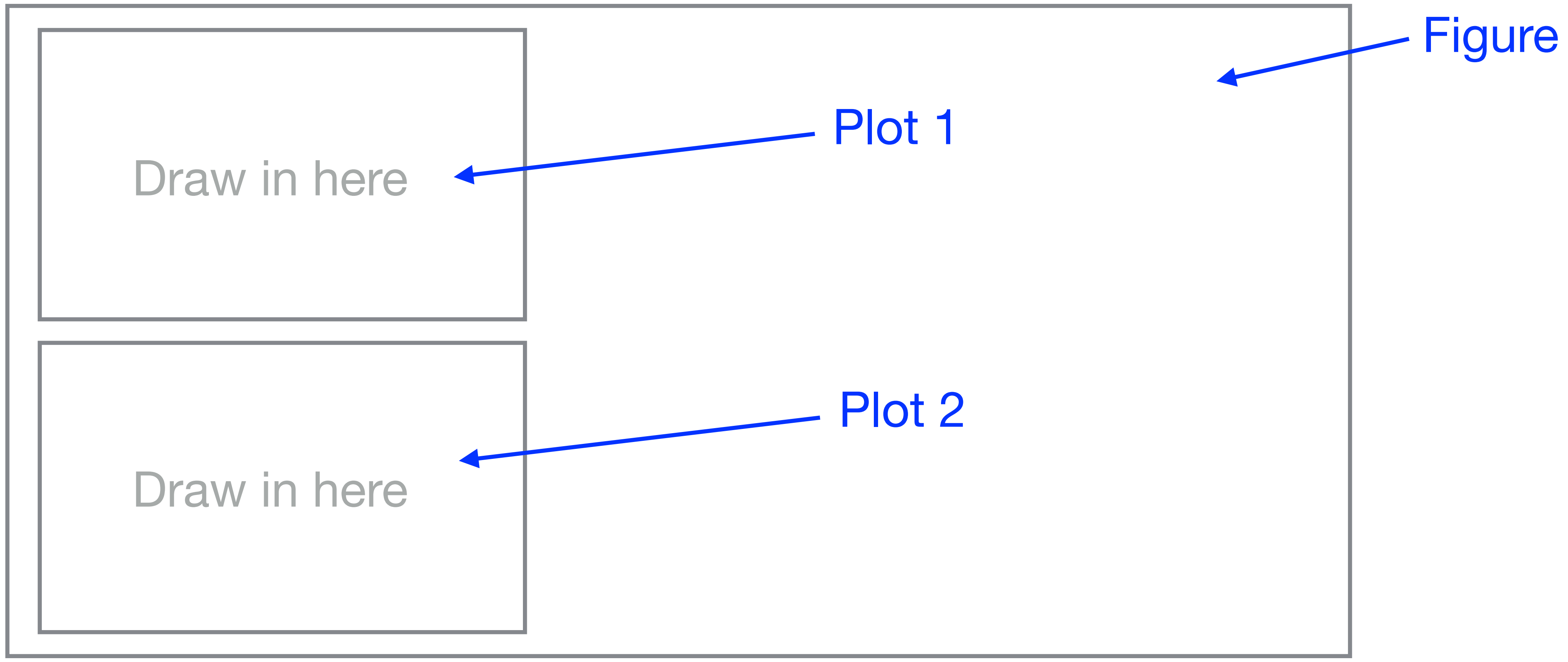
Import the required packages

```
%matplotlib inline
```

Need to use a "magic command" to tell the notebook to render the plot inside the notebook. Otherwise it will not appear!

# Visualisation with Matplotlib

- The main visualisation area in Matplotlib is a figure. This figure can contain one or more plots.



- Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels etc.
- States are preserved across function calls, so that it keeps track of things like the current figure and plotting area.

# Simple Plots

- A simple plot can be used to show the values in a standard Python list. The values in the list are plotted on the y-axis.
- Matplotlib assumes the values are a sequence and automatically assigns values (0,1,2,...) to the x-axis.

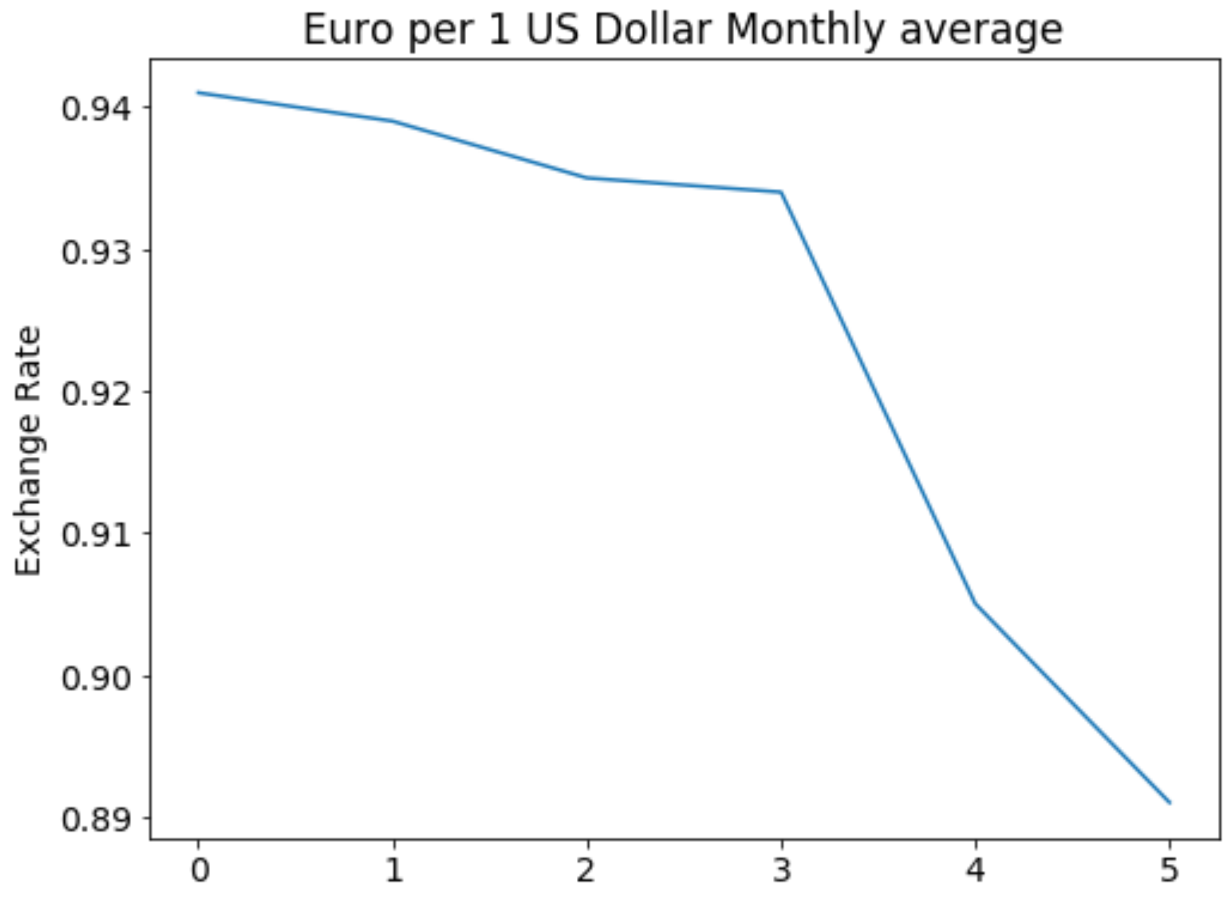
```
data = [0.941,0.939,0.935,
0.934,0.905,0.891]
```

Create a new figure

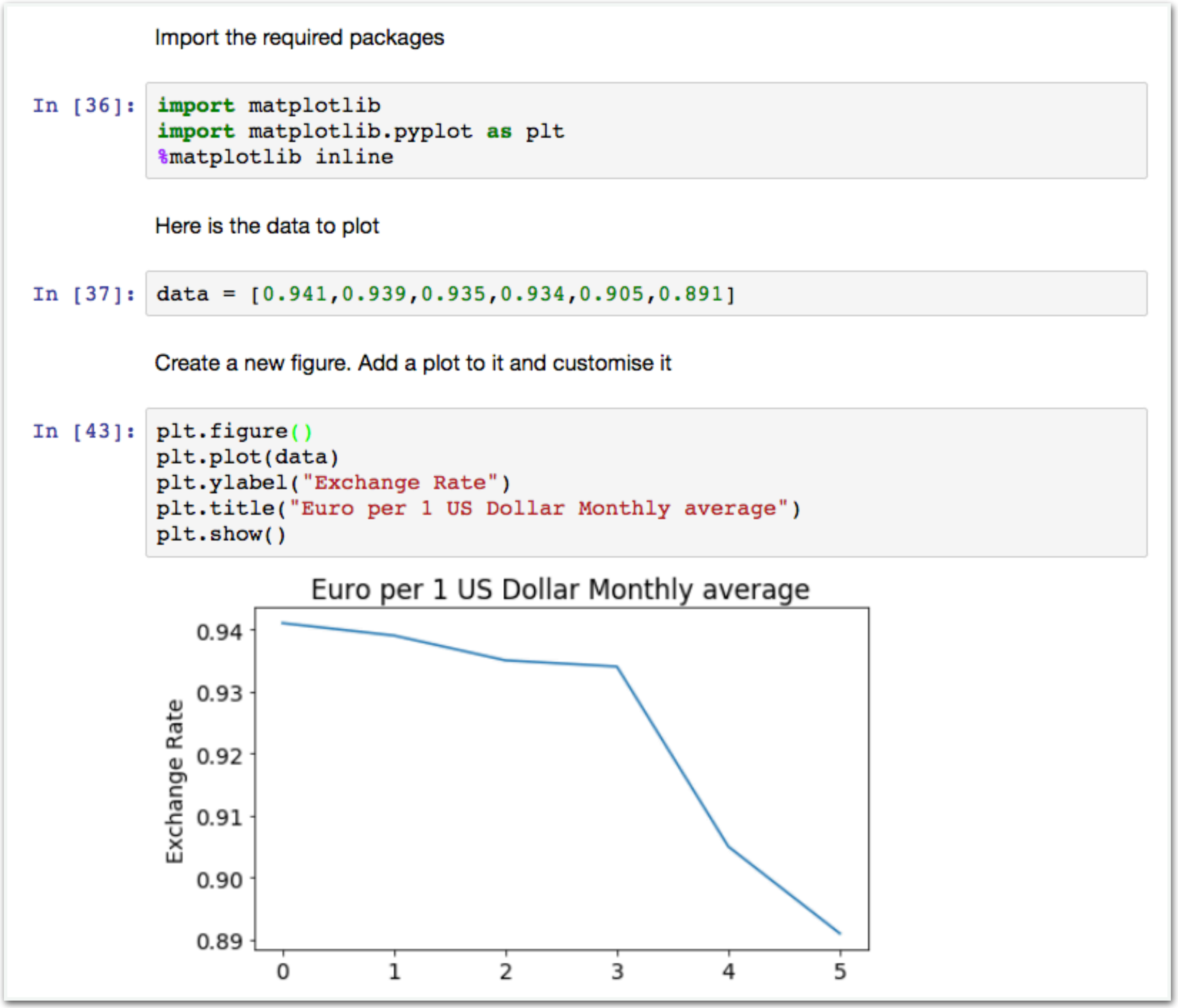
```
plt.figure()
```

Add a plot to the figure, and customise it

```
plt.plot(data)
plt.ylabel("Exchange Rate")
plt.title("Euro per 1 US Dollar")
plt.show()
```



# Simple Plot in a Notebook



Import the required packages

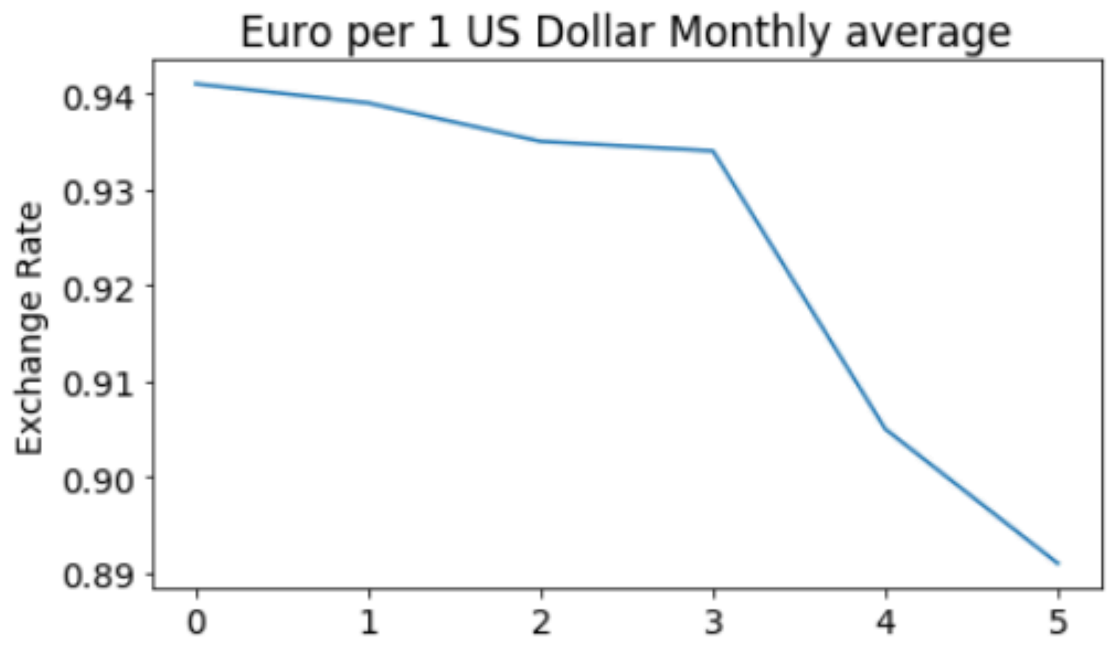
```
In [36]: import matplotlib
         import matplotlib.pyplot as plt
         %matplotlib inline
```

Here is the data to plot

```
In [37]: data = [0.941,0.939,0.935,0.934,0.905,0.891]
```

Create a new figure. Add a plot to it and customise it

```
In [43]: plt.figure()
         plt.plot(data)
         plt.ylabel("Exchange Rate")
         plt.title("Euro per 1 US Dollar Monthly average")
         plt.show()
```



# Matplotlib Scatter Plots

- In a scatter plot, the values of two variables are plotted along two axes (X and Y). The pattern of the points reveals if there is any correlation present between the two variables.

```
import matplotlib.pyplot as plt
Import matplotlib
%matplotlib inline
```

Populate our data - 6 pairs of X and Y coordinates

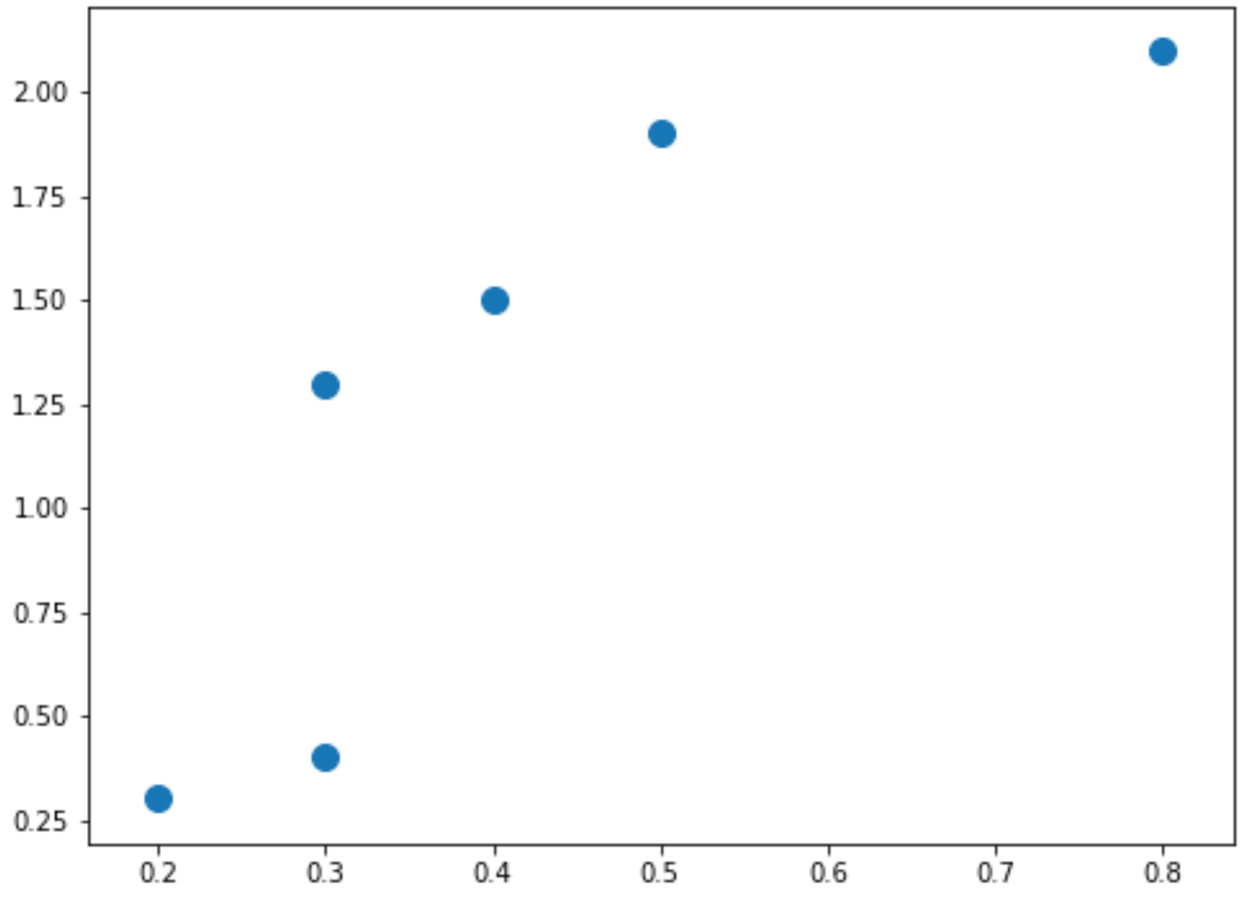
```
x = [0.4, 0.3, 0.5, 0.8, 0.2, 0.3]
y = [1.5, 1.3, 1.9, 2.1, 0.3, 0.4]
```

Create a new figure

```
plt.figure()
```

Draw scatter plot using the X & Y data

```
matplotlib.pyplot.scatter( x, y )
```



Scatter plot. X axis is horizontal, Y axis is vertical.

## Matplotlib Scatter Plots

- Matplotlib plots can be customised in many different ways. For examples see: https://matplotlib.org/api

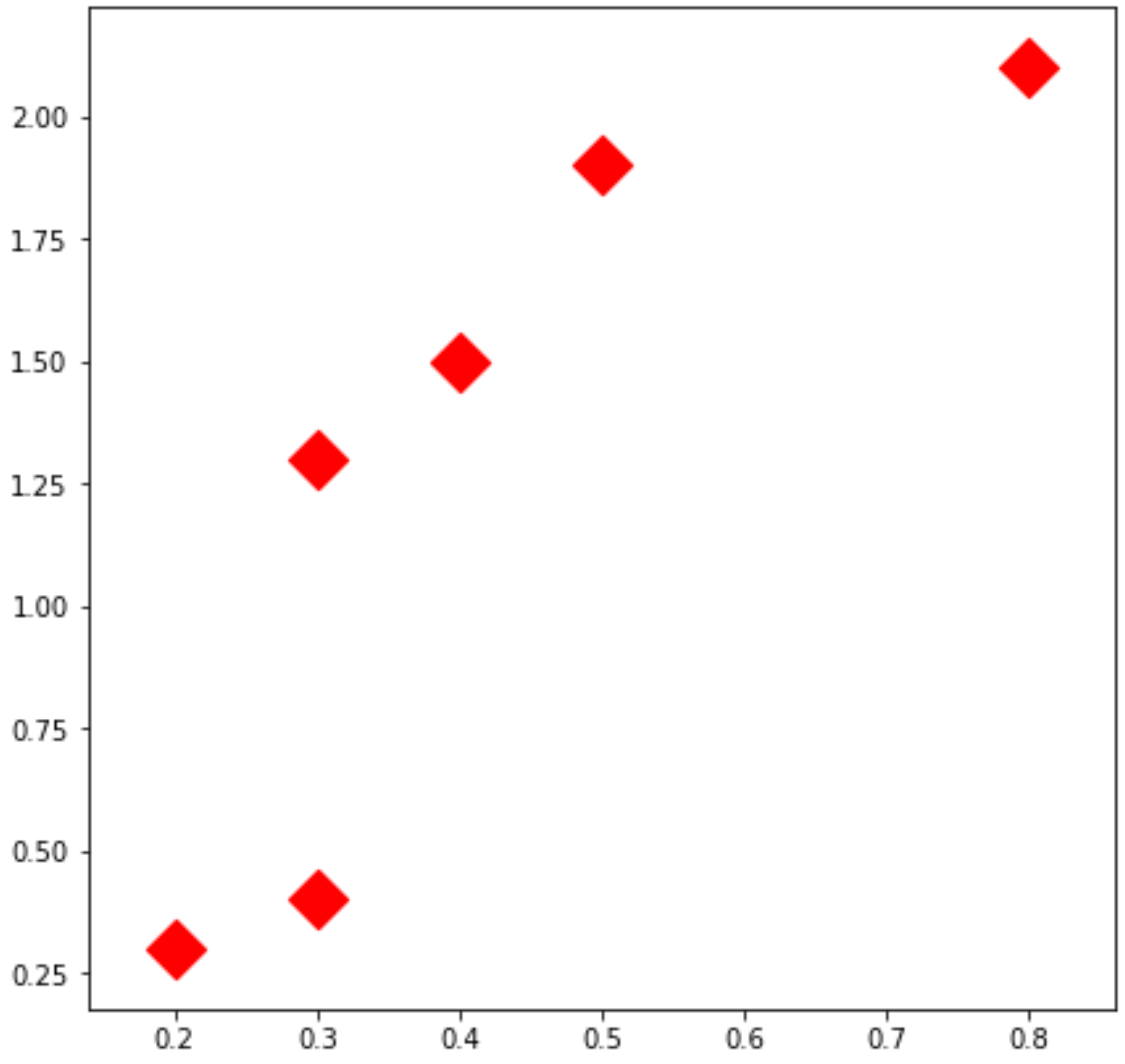
```
x = [0.4, 0.3, 0.5, 0.8, 0.2, 0.3]
y = [1.5, 1.3, 1.9, 2.1, 0.3, 0.4]
```

Specify the size of the figure - i.e. its width and height as a tuple.

```
plt.figure(figsize=(7,7))
```

Specify the size (s) of the points, the colour (c) of the points, and the shape of the marker to use for points

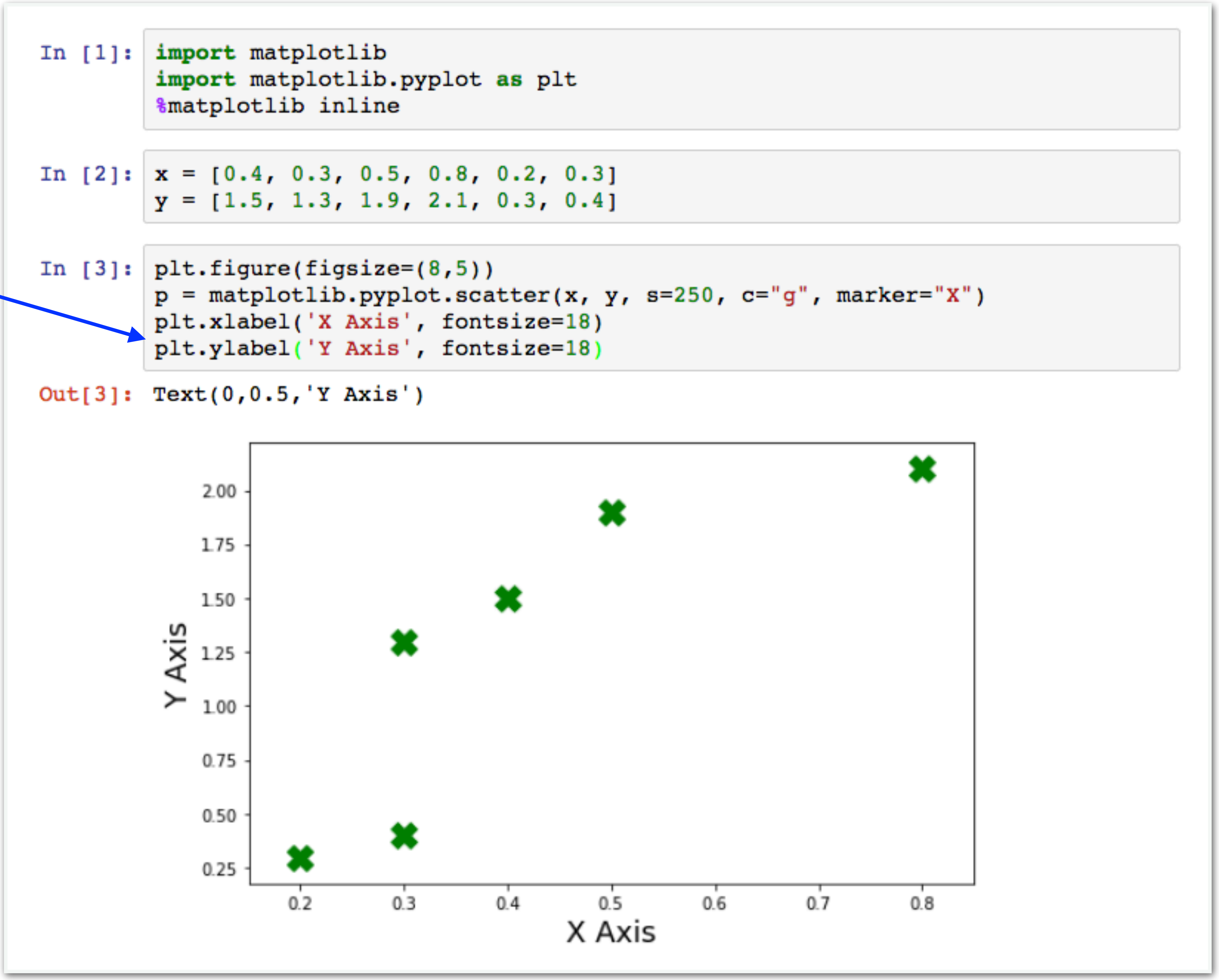
```
matplotlib.pyplot.scatter(x, y,
s=250, c="r", marker="D")
```



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

## Matplotlib Scatter Plots

Create plot, then add text labels to the X and Y axis



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

```
In [2]: x = [0.4, 0.3, 0.5, 0.8, 0.2, 0.3]
        y = [1.5, 1.3, 1.9, 2.1, 0.3, 0.4]
```

```
In [3]: plt.figure(figsize=(8,5))
        p = matplotlib.pyplot.scatter(x, y, s=250, c="g", marker="X")
        plt.xlabel('X Axis', fontsize=18)
        plt.ylabel('Y Axis', fontsize=18)

Out[3]: Text(0,0.5,'Y Axis')
```

# Matplotlib Pie Charts

- Various other charts are available in Matplotlib, e.g. pie charts:

Create the data

```
pops = [1100, 198, 91, 76]
cities = ["Dublin","Cork","Limerick","Galway"]
```

Create a new figure

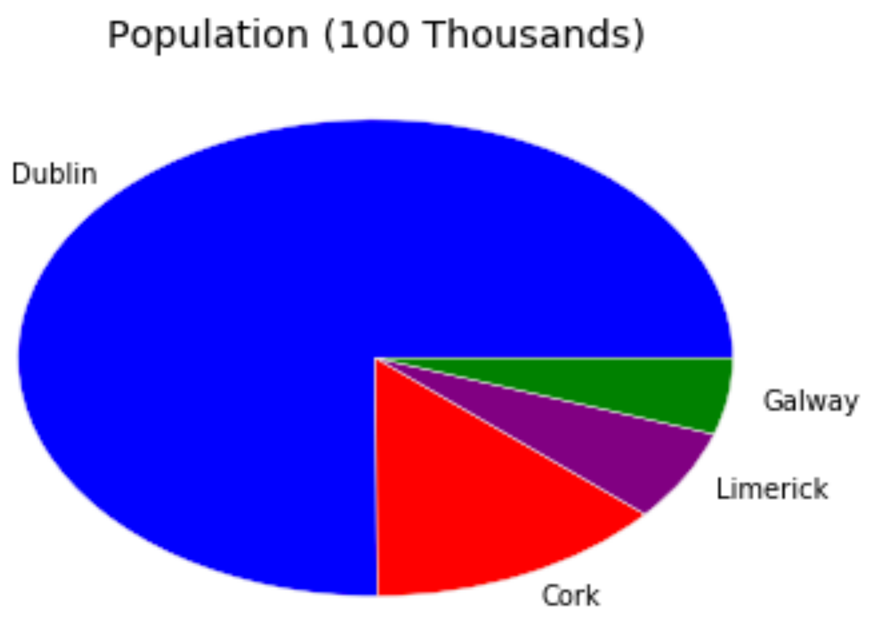
```
plt.figure()
```

Create a pie chart with specified values, wedge labels, and wedge colours.

```
mycolors=["blue","red","purple","green"]
p = plt.pie(pops, labels=cities,
colors=mycolors)
```

Add a title to the plot

```
plt.title("Population (100 Thousands)")
```



## Matplotlib Bar Charts

- We could represent the same data using a bar chart:

Create a new figure

```
plt.figure()
```

Need to specify y-axis positions too

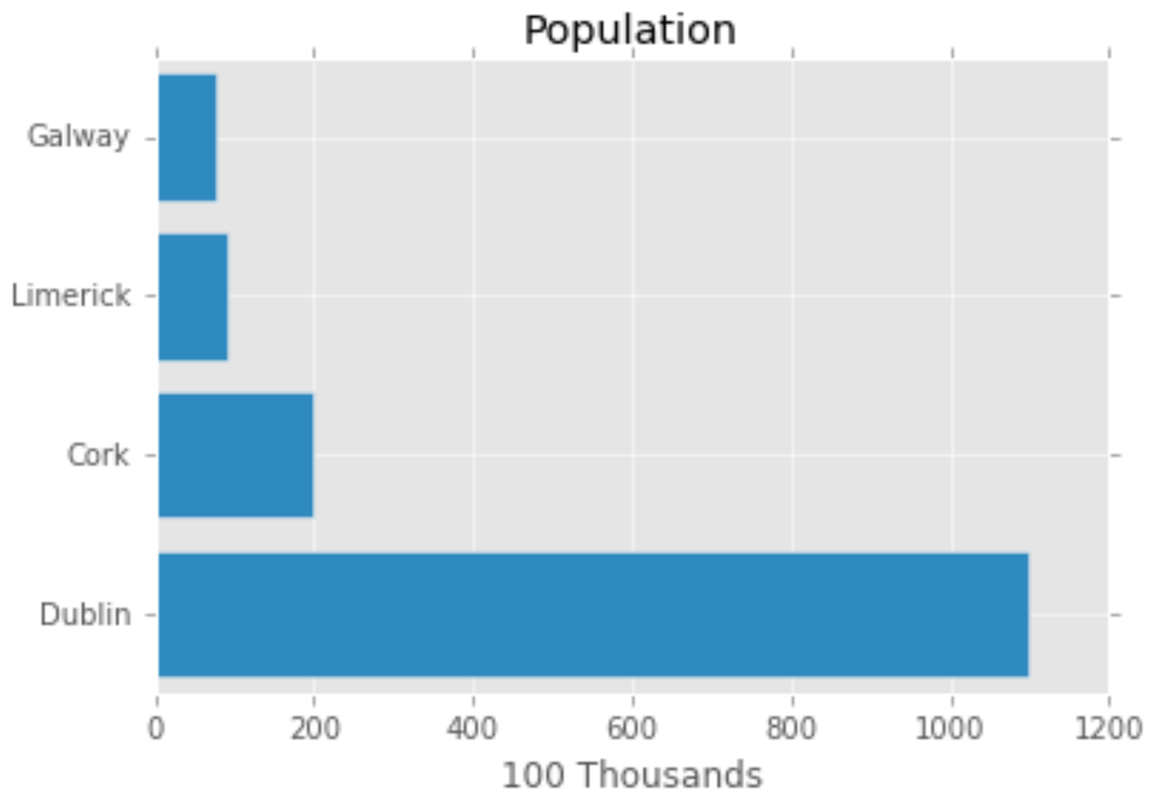
```
y_pos = [0, 1, 2, 3]
```

Create bar chart, with labels

```
plt.barh(y_pos, pops, align='center')
plt.yticks(y_pos, cities)
```

Add an axis label and title to the chart

```
plt.xlabel("100 Thousands")
plt.title("Population")
```



# Plotting with Pandas

- Pandas provides basic visualisation functionality that uses Matplotlib under the hood, but with a simpler interface based around Series and Data Frames using the `plot()` function.

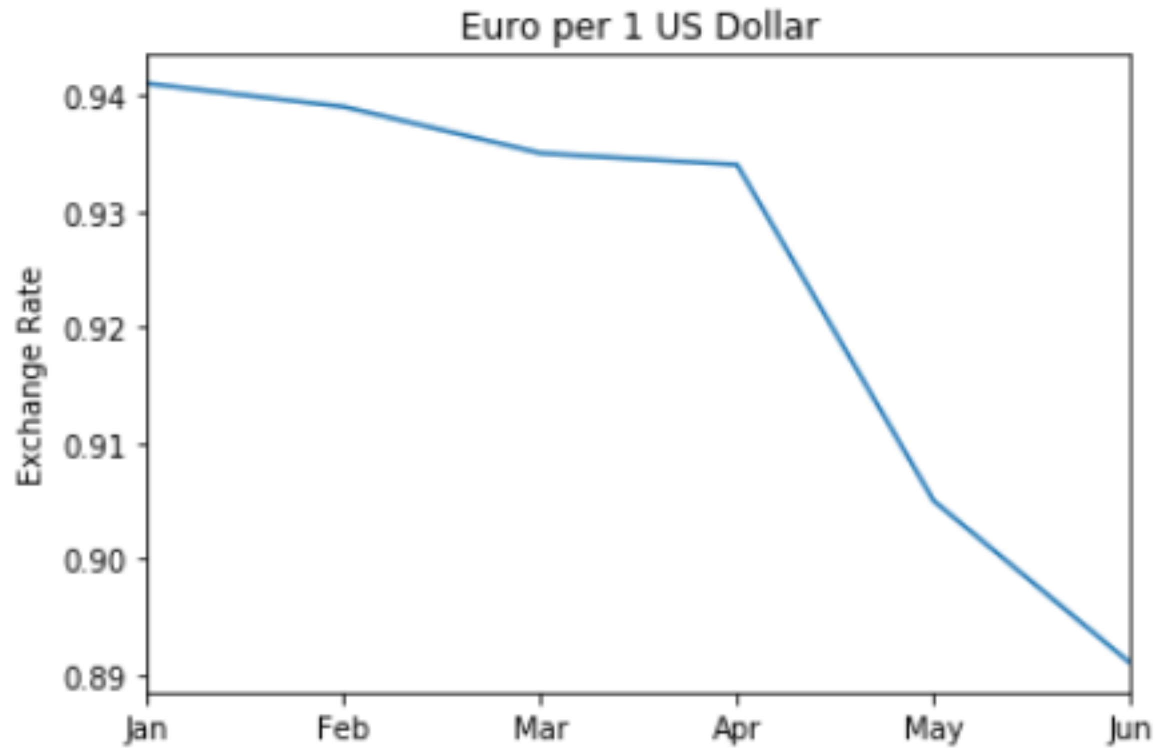
Import Pandas, create a Series

```
import pandas as pd
%matplotlib inline
data = [0.941,0.939,0.935, 0.934,0.905,0.891]
labels = ["Jan","Feb","Mar","Apr","May","Jun"]
exchange = pd.Series( data, labels )
```

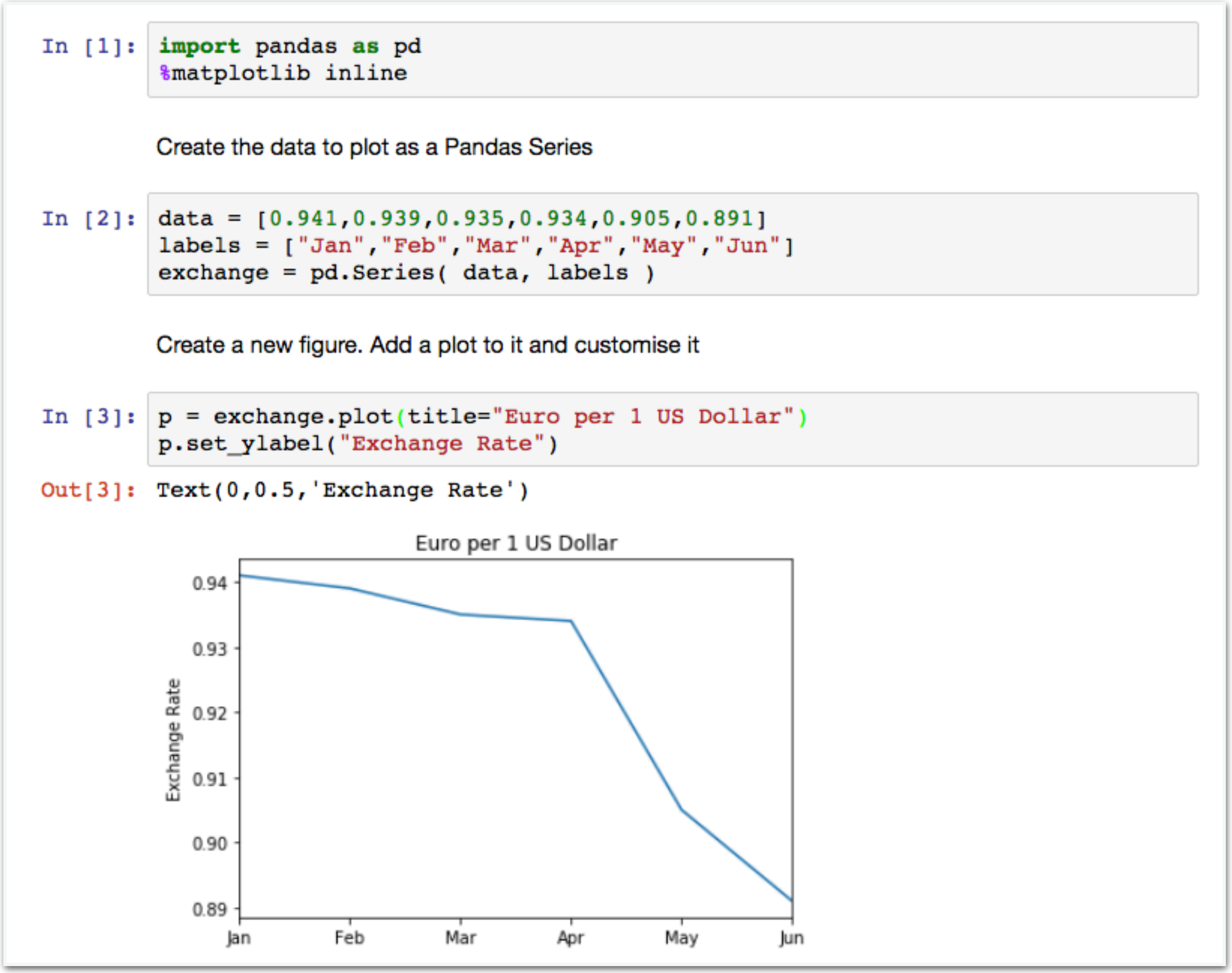
Create line chart from the Series using `plot()`, and customise it

```
p = exchange.plot(title="Euro per 1 US Dollar")
p.set_ylabel("Exchange Rate")
```

The figure will get created and displayed automatically.



# Series Plot in a Notebook



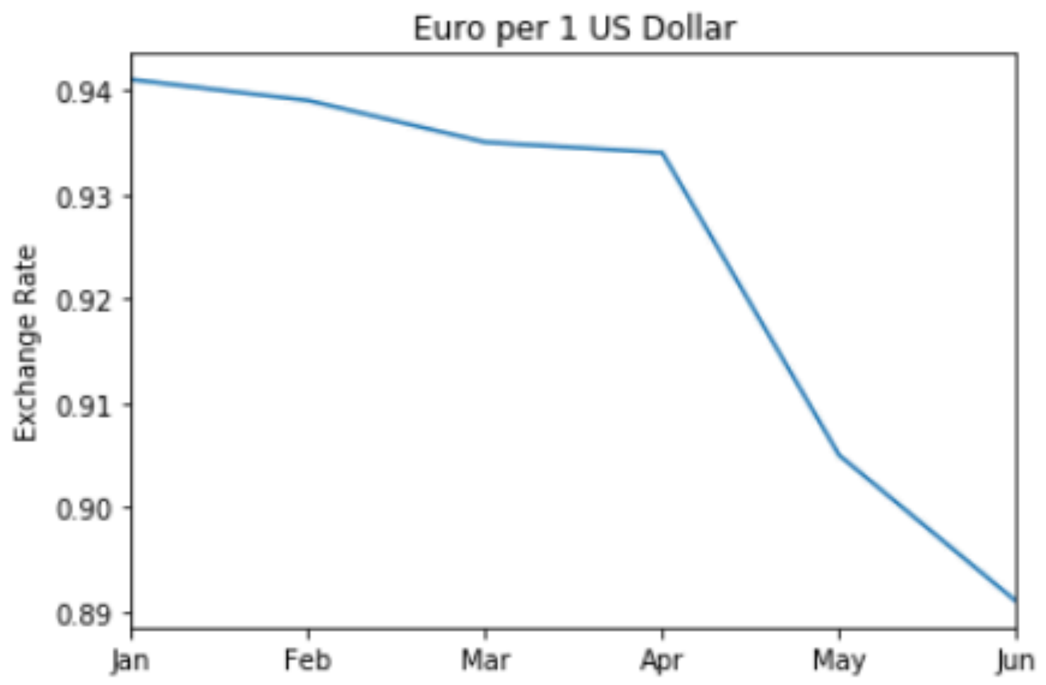
```
In [1]: import pandas as pd
        %matplotlib inline
```

Create the data to plot as a Pandas Series

```
In [2]: data = [0.941,0.939,0.935,0.934,0.905,0.891]
        labels = ["Jan","Feb","Mar","Apr","May","Jun"]
        exchange = pd.Series( data, labels )
```

Create a new figure. Add a plot to it and customise it

```
In [3]: p = exchange.plot(title="Euro per 1 US Dollar")
        p.set_ylabel("Exchange Rate")

Out[3]: Text(0,0.5,'Exchange Rate')
```



# Plotting with Pandas

- If our data is stored in a Data Frame, we can select individual columns to plot, by specifying the column's index or position.

| Month | Sales | Units |
|---|---|---|
| jan | 20392 | 199 |
| feb | 24221 | 239 |
| mar | 26773 | 255 |
| apr | 40188 | 405 |
| may | 41972 | 420 |
| jun | 36375 | 354 |
| jul | 31228 | 299 |
| aug | 41368 | 396 |
| sep | 55227 | 552 |
| oct | 42341 | 400 |
| nov | 48585 | 421 |
| dec | 60721 | 583 |

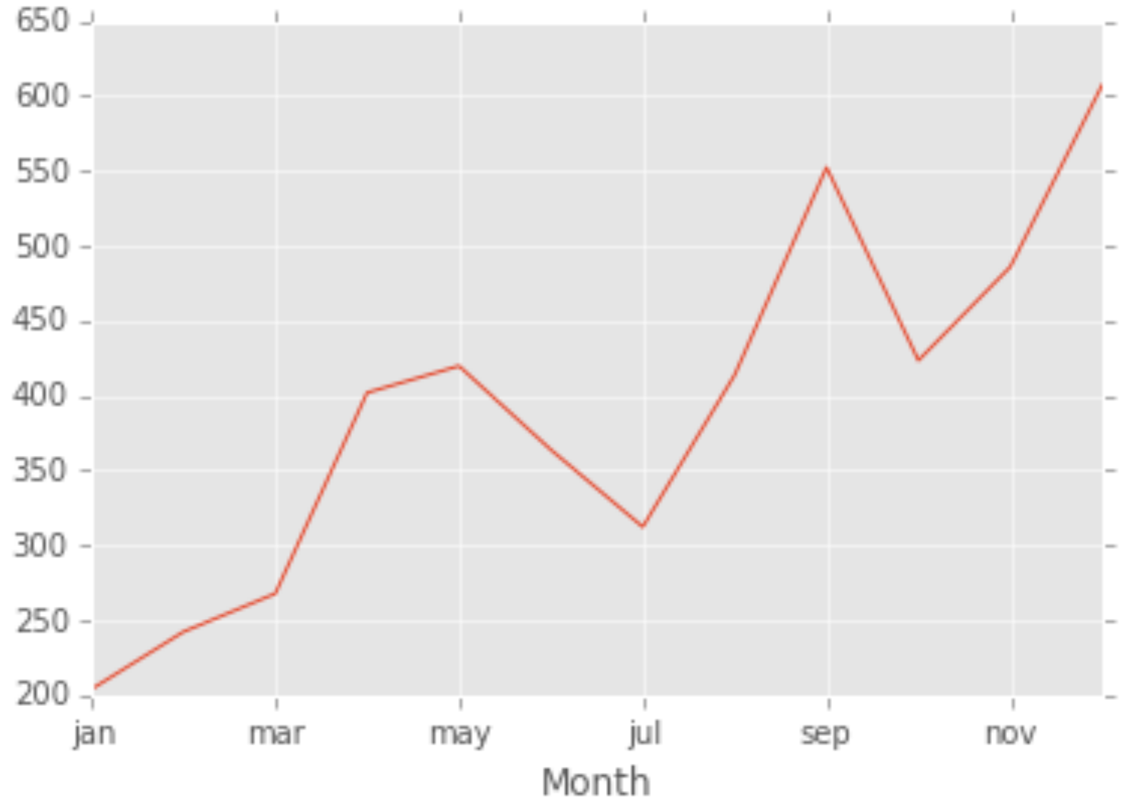
Import Pandas, load an existing CSV file as a Data Frame, and set the row index to be "Month".

```
import pandas as pd
%matplotlib inline
df = pd.read_csv("sales1.csv",index_col="Month")
```

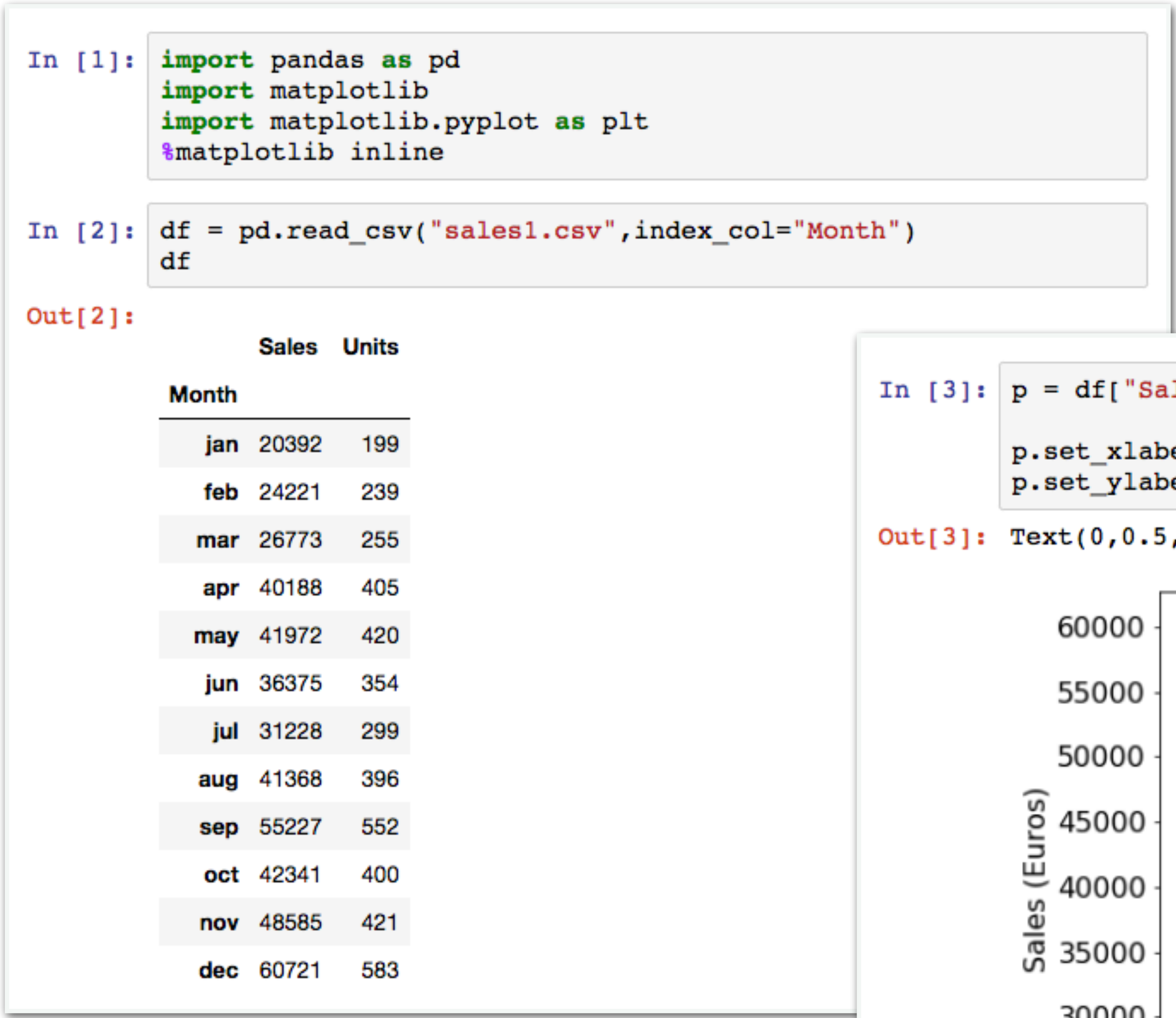
Create a line chart from the specified column "Sales" using the `plot()` function.

```
p = df["Sales"].plot()
```

The Y axis corresponds to the column "Sales", and the X axis is the row index.



# Data Frame Plot in a Notebook



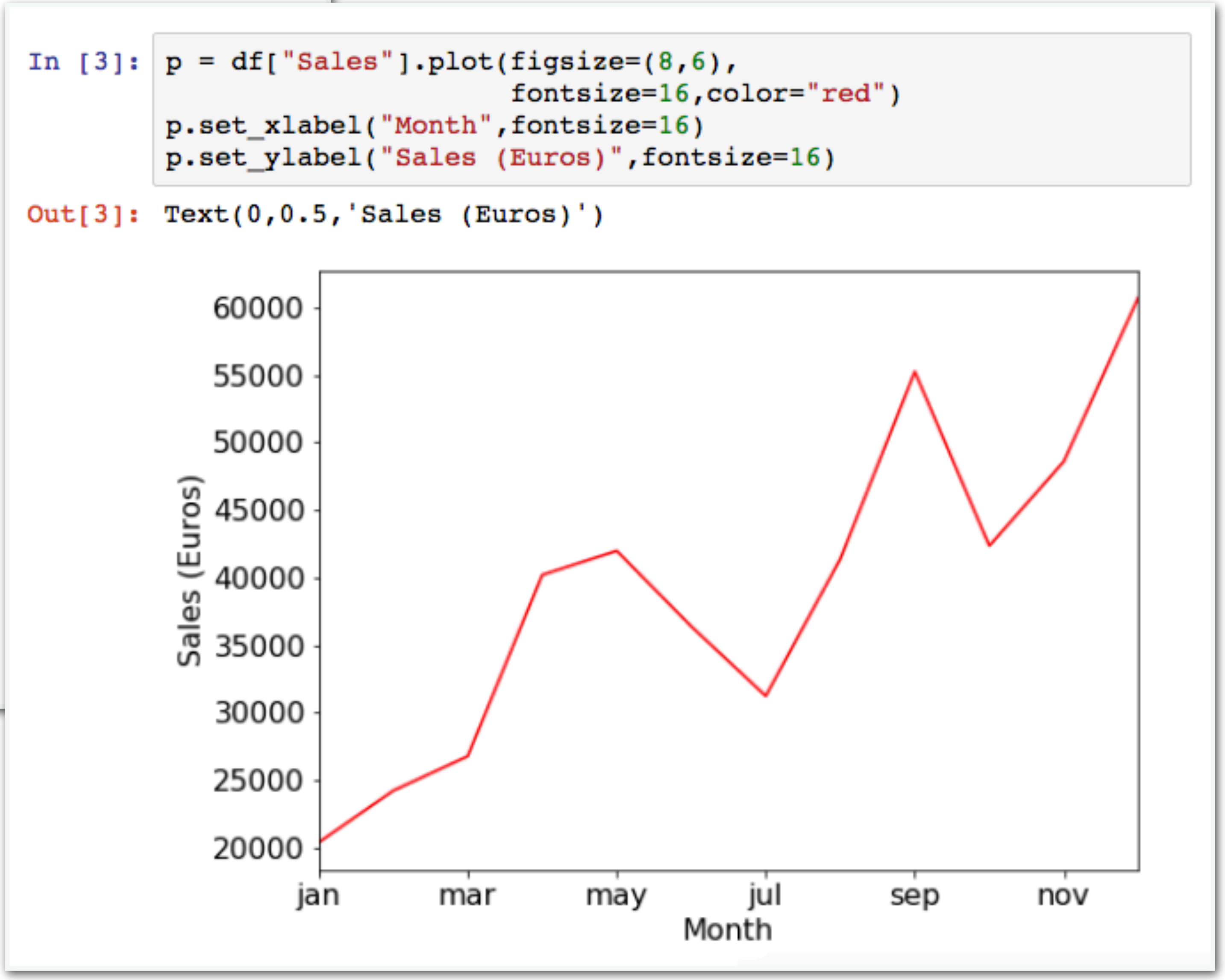
```
In [1]: import pandas as pd
        import matplotlib
        import matplotlib.pyplot as plt
        %matplotlib inline
```

```
In [2]: df = pd.read_csv("sales1.csv",index_col="Month")
        df
```

Out[2]:

| Month | Sales | Units |
|---|---|---|
| jan | 20392 | 199 |
| feb | 24221 | 239 |
| mar | 26773 | 255 |
| apr | 40188 | 405 |
| may | 41972 | 420 |
| jun | 36375 | 354 |
| jul | 31228 | 299 |
| aug | 41368 | 396 |
| sep | 55227 | 552 |
| oct | 42341 | 400 |
| nov | 48585 | 421 |
| dec | 60721 | 583 |

Import Pandas, load an existing CSV file as a Data Frame, set the row index to be "Month", and inspect the data.



```
In [3]: p = df["Sales"].plot(figsize=(8,6),
                             fontsize=16,color="red")
        p.set_xlabel("Month",fontsize=16)
        p.set_ylabel("Sales (Euros)",fontsize=16)
```

```
Out[3]: Text(0,0.5,'Sales (Euros)')
```

Plot the column "Sales", and customise the plot appearance, and the axis labels

# Plotting with Pandas

- A Pandas Series can be plotted using bar charts or pie charts. Example: Iris Dataset

```
import pandas as pd
data = pd.read_csv("iris.csv")
```

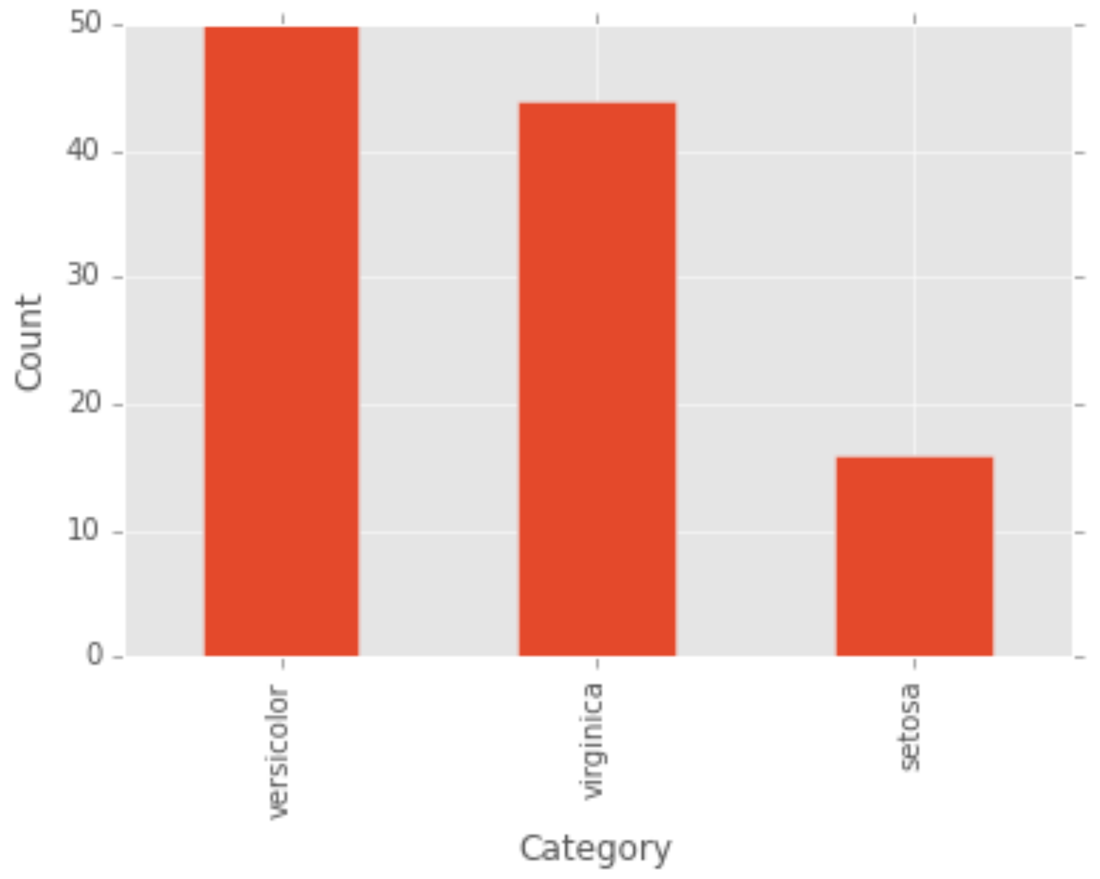
|   | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| **0** | 5.7 | 2.8 | 4.1 | 1.3 | versicolor |
| **1** | 7.0 | 3.2 | 4.7 | 1.4 | versicolor |
| **2** | 5.8 | 2.7 | 3.9 | 1.2 | versicolor |
| **3** | 5.5 | 2.5 | 4.0 | 1.3 | versicolor |
| **4** | 7.7 | 3.0 | 6.1 | 2.3 | virginica |

```
counts = data["species"].value_counts()
```

```
versicolor    50
virginica     44
setosa        16
```

Create bar chart from the Series

```
p = counts.plot(kind="bar")
p.set_xlabel("Category")
p.set_ylabel("Count")
```



Save the figure as a PNG image

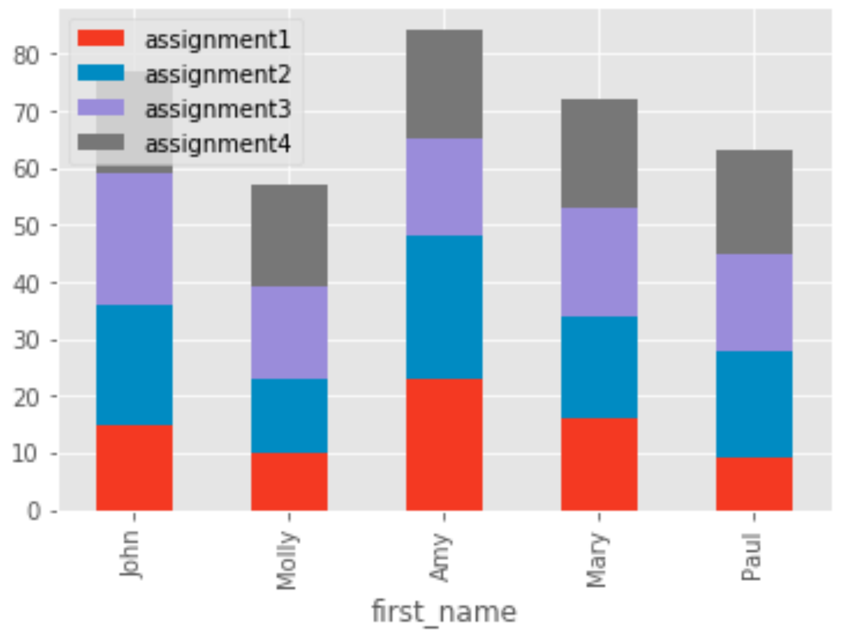
```
fig = p.get_figure()
fig.savefig("iris-bars.png")
```

## Plotting with Pandas

- We can also generate stacked bar charts in a similar way, based on multiple columns in a Data Frame.

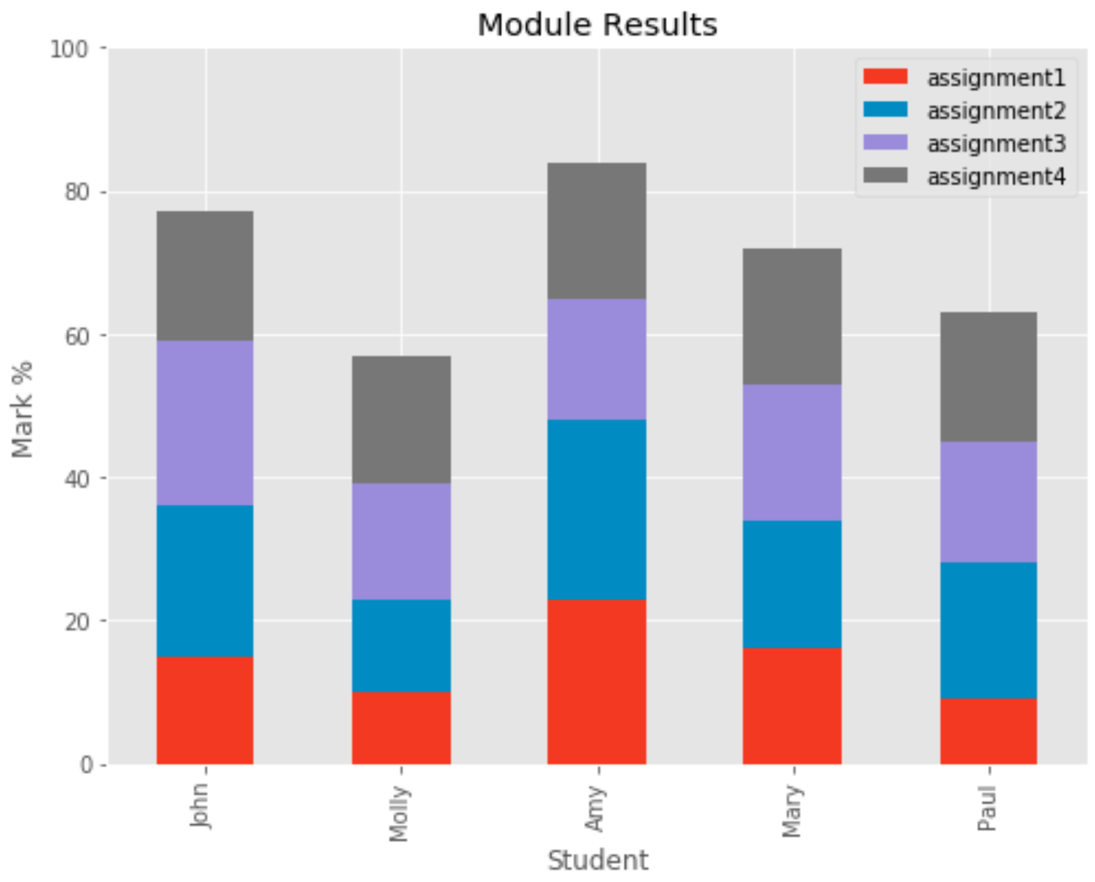
| | assignment1 | assignment2 | assignment3 | assignment4 |
|---|---|---|---|---|
| **first_name** | | | | |
| **John** | 15 | 21 | 23 | 18 |
| **Molly** | 10 | 13 | 16 | 18 |
| **Amy** | 23 | 25 | 17 | 19 |
| **Mary** | 16 | 18 | 19 | 19 |
| **Paul** | 9 | 19 | 17 | 18 |

```
p = df.plot.bar(stacked=True)
```



Customise plot size, axes, and title

```
p = df.plot.bar(stacked=True,figsize=(10, 6))
p.set_ylim((0,100))
p.set_xlabel("Student")
p.set_ylabel("Mark")
p.set_title("Module Results")
```

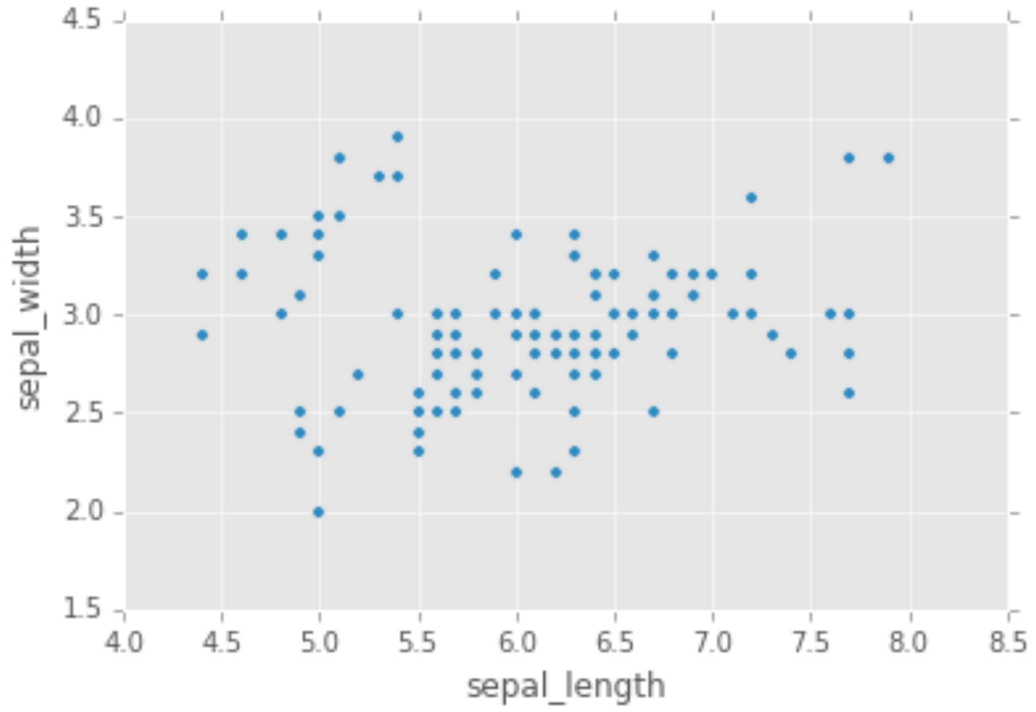


# Plotting with Pandas

- We can use XY scatter plots to visualise the relationship between pairs of columns in a Data Frame.

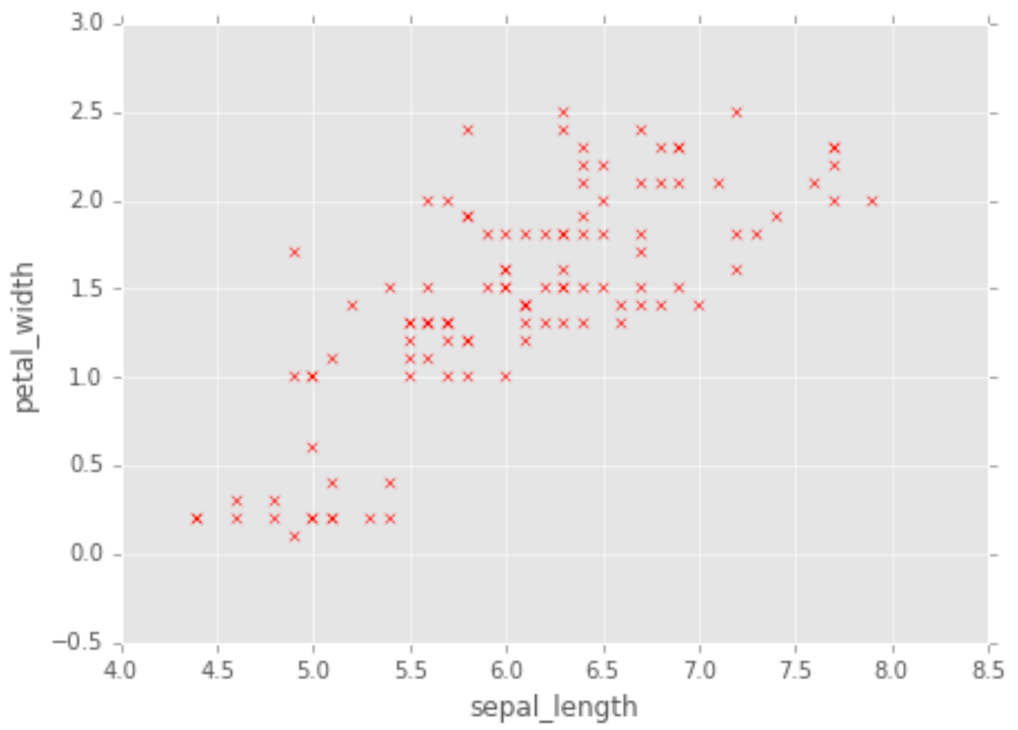
Basic XY scatter for 2 columns

```
data.plot(kind="scatter",
x="sepal_length",
y="sepal_width")
```



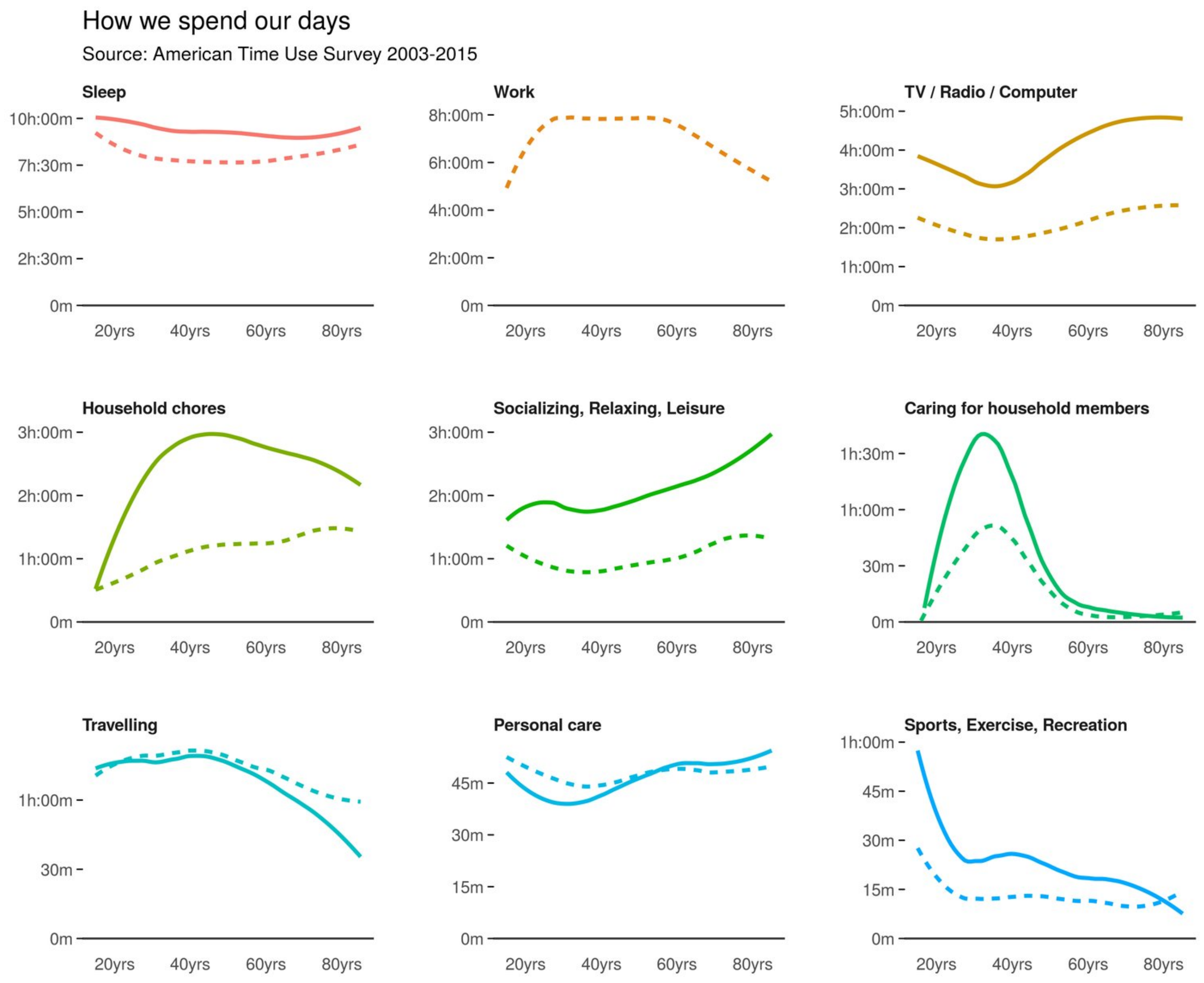
Customise the appearance of the plot by changing the appearance of the points and plot size

```
data.plot(kind="scatter",
x="sepal_length",
y="petal_width",
marker='x',
color='red',
figsize=(7, 5))
```



# Small Multiples

- Small multiples involves using many plots using the exact same axes, allowing reader to easily discover patterns by eye.

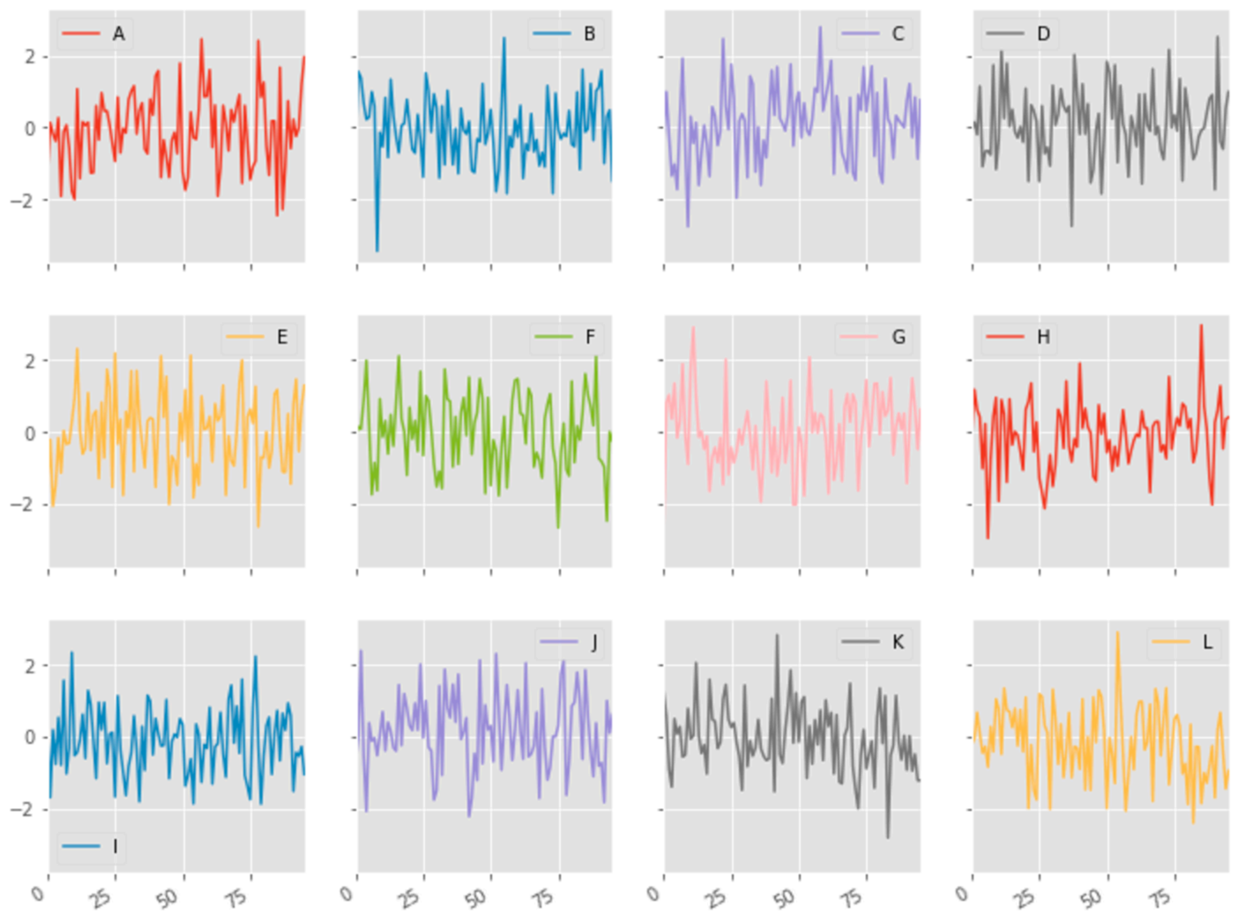


https://www.reddit.com/r/dataisbeautiful

# Small Multiples with Pandas

- We can display each series on a separate subplot (i.e. "small multiples") by setting `subplots=True`.
- Example: For a Data Frame with 12 columns (A to L). Use subplots for each column, arranged as 3 rows and 4 columns

```
df.plot(kind='line', subplots=True, layout=(3,4), figsize=(12,10), sharey=True)
```



Make sure that all Y-axis limits are the same, so that comparisons can be made.

## Small Multiples with Pandas

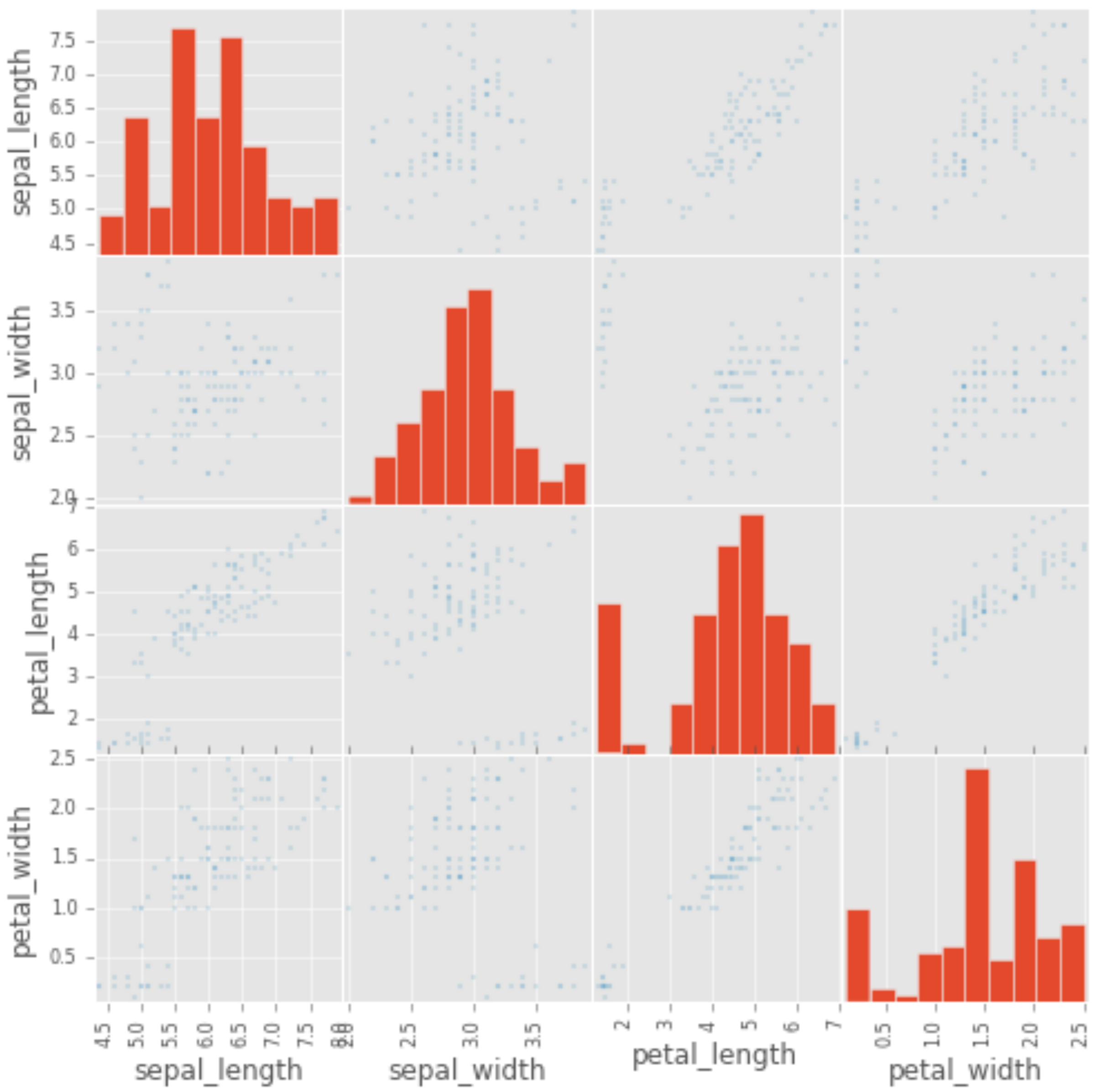
- We can visualise the relationship between all pairs of columns using a "small multiples" approach, via a scatter matrix:

```
from pandas.plotting import scatter_matrix
```

```
scatter_matrix(data)
```

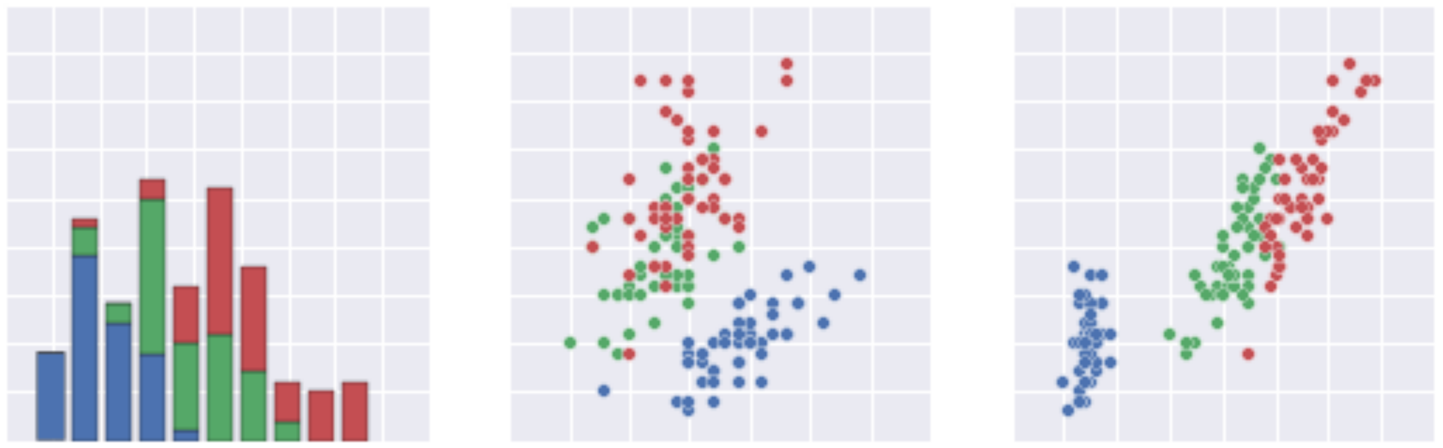
Output is a column-by-column matrix of XY scatter plots off the diagonal.

The plots on the diagonal show the distribution of values for each of the individual features.

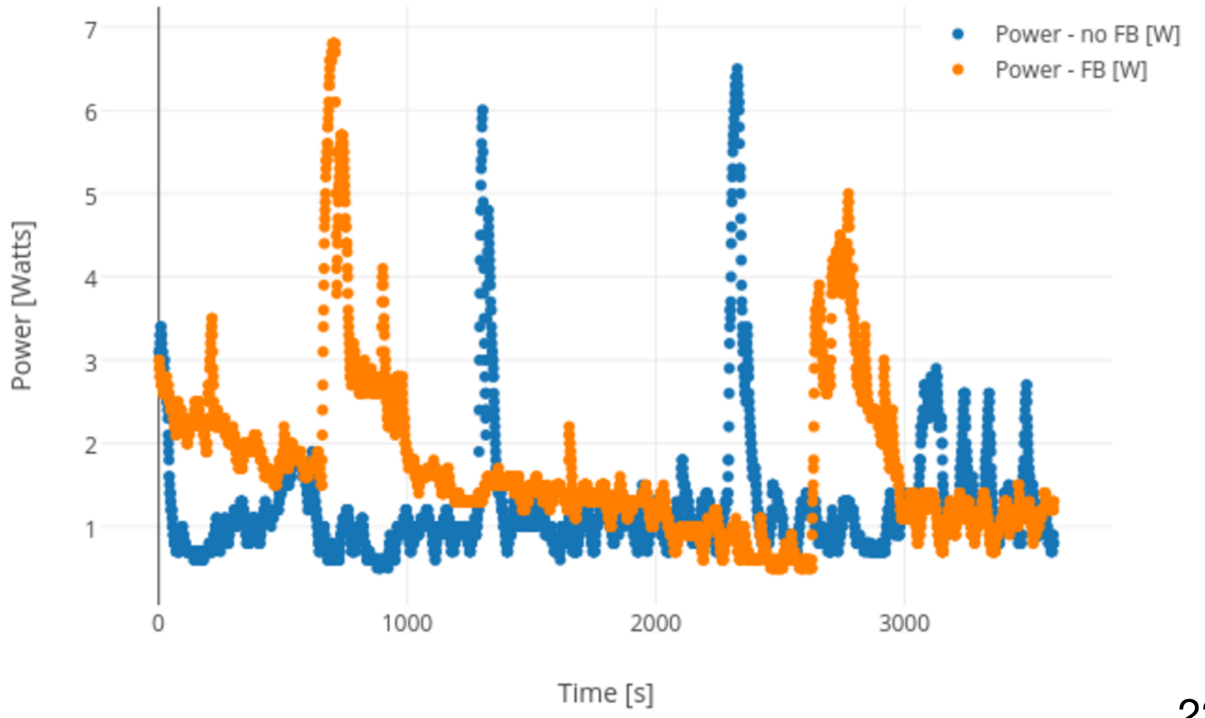


## Other Visualisation Packages

- **Seaborn**: A Python visualisation library based on Matplotlib which provides a higher level interface for drawing attractive statistical graphics. (https://stanford.edu/~mwaskom/software/seaborn/)

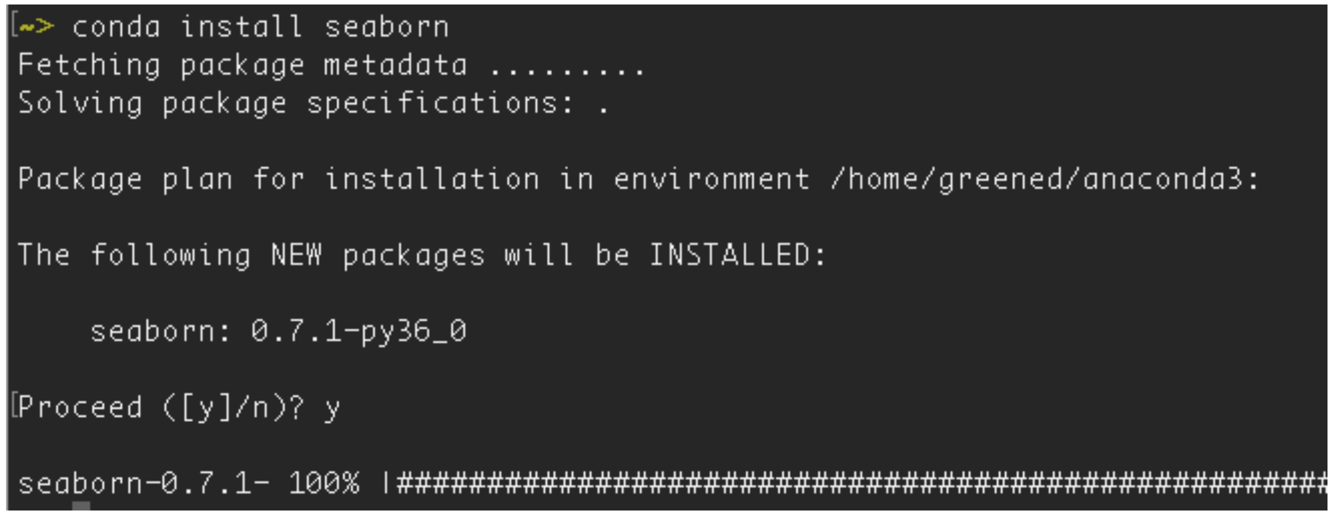


- **Plotly**: An online analytics and data visualisation tool. Plotly's Python package can be used to make interactive graphs directly from Pandas Data Frames. (https://plot.ly/pandas)



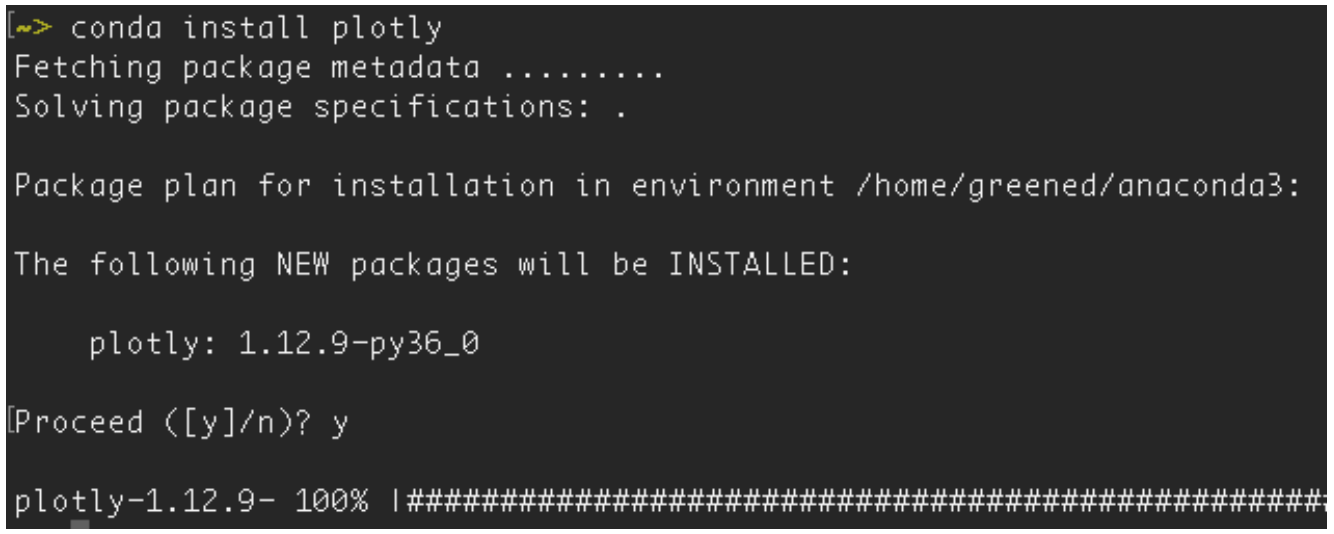
## Other Visualisation Packages

- To install third party packages, run the conda tool at the terminal/command line (not in Python).
- Run: `conda install <package_name>`



```
~> conda install seaborn
Fetching package metadata .........
Solving package specifications: .

Package plan for installation in environment /home/greened/anaconda3:

The following NEW packages will be INSTALLED:

    seaborn: 0.7.1-py36_0

Proceed ([y]/n)? y

seaborn-0.7.1- 100% |####################################################
```

Install the Seaborn package and any dependencies.



```
~> conda install plotly
Fetching package metadata .........
Solving package specifications: .

Package plan for installation in environment /home/greened/anaconda3:

The following NEW packages will be INSTALLED:

    plotly: 1.12.9-py36_0

Proceed ([y]/n)? y

plotly-1.12.9- 100% |####################################################
```

Install the Plotly package and any dependencies.