# Intro to Notebooks, Modules & Matplotlib

Dr Philip Lewis

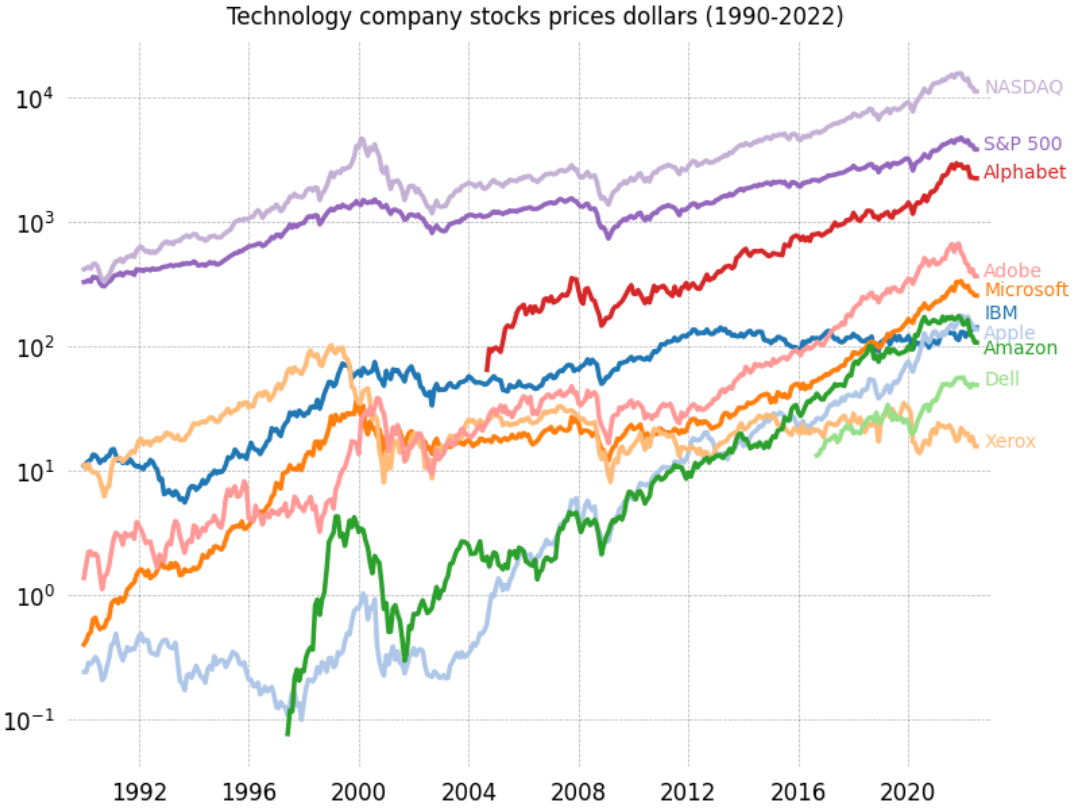
University of Exeter

# Aims

- How to use Jupyter Notebooks for coding
- Introduce Python coding terminology:
  (functions, objects, methods, modules)
- Demonstrate use of example Python modules
- Learn how to make plots with Matplotlib

University of Exeter

# Target




Adapted from: https://matplotlib.org/stable/gallery/showcase/stock_prices.html

University of Exeter

# Topics

1. JupyterLab and Python Notebooks
2. Python coding: Functions Objects, Methods & Modules
3. Example useful modules
4. Making a Plot with Matplotlib
5. Customising Plots with Matplotlib
6. Plot Types in Matplotlib

University *of* Exeter

# Python Notebooks and JupyterLab

Dr Philip Lewis

University of Exeter

# Python Notebooks

- A modern code file format that allows us to store:
  **code & output / formatted text / tables / figures**
- Can include text sections (in markdown format)
- Based on web technology and developed by Jupyter
- `.ipynb` file extension (**i**nteractive **py**thon **n**ote**b**ook)

University *of* Exeter

# JupyterLab

- A complete code IDE based on browser technology
- Includes notebook editor, .py editor, code terminal, file browser and debugger
- succeeded Jupyter Notebook (which only allowed notebook editing).
- Runs in a web browser window (so has some differences to normal applications).

Install and run via Anaconda Navigator or JupyterLab Desktop
You can also setup within VS Code (Can be tricky)

University of Exeter

# Working in a Notebook

**Please watch the separate demo video!**

University of Exeter

# Key differences .py vs .ipynb

- In notebooks code cells can be run individually (and out-of-sequence!).
- If you encounter odd behaviour use **Run All** to run all code cells in sequence.
- Notebooks are designed for interactive coding. Some outputs e.g. figures, and output from last line in code cells are automatically displayed

In **.py** code files we always need to use:

| | |
|---|---|
| `print()` | for text |
| `plt.show()` | for figures |

University *of* Exeter

# Python tools:

# Functions, objects, methods & modules

Dr Philip Lewis

University of Exeter

# Built-in Functions

- Python contains a set of useful tools in its standard installation.
- These include in-built **functions** such as:

```
print()        len()
sorted()       range()
sum()          min() , max()
```

University of Exeter

# Built-in Functions

- a function is a piece of code that does a particular job. (We will see in future sessions how we can write our own functions).
- we run a function by calling it, using the function name followed by round brackets ()
- the brackets hold any input information (arguments) the function needs to run
- commas are used separate values if multiple values are passed.

e.g. `len()` is used to find the length of a list

e.g. exit() can be used to exit an interactive Python session.

e.g. the argument for print( ___ ) is the string or value to display

e.g. range(3,10) can take a lower and upper limit

University of Exeter

# Built-in Functions

Python has conversion functions such as:

```
int()   float()   str()
```

These can convert between data types e.g.

```
myval = 4.1
print( int(myval) )
# 4
```

University of Exeter

# Built-in Functions

We can check the type of a variable using the `type()` function.

```
myval = 4.1

print( type(myval) )

# <class 'float'>
```

University of Exeter

# Built-in Functions

We can round a floating point number using the `round()` function.

```
myval = 5.1234

print( round(myval, 2) )

# 5.12
```

University of Exeter

# Built-in Functions

`abs()` can be used to find the absolute size of a value. It converts a negative into a positive value.

```
x = abs(-6.5)
# x is 6.5
```

`pow()` can be used as an alternative to the ** operator to raise one number to the power of another.

```
y = pow(3,2)
# y is 9
```

University of Exeter

# Objects and Methods

The `sorted`, `reverse`, `min`, `max` and `sum` built-in functions operate on a list, e.g.

```
x = [ 2, 7, 9, 1 ]

sum(x)

# 19
```

However we have also seen special functions that act on lists.

Functions that act on a Python object are called methods.

University of Exeter

# Objects and Methods

An object method is called using the following notation:

*`object.method()`*

For example the `.sort()` method can be called on any list object.

```
x = [ 2, 7, 9, 1 ]
x.sort()
print(x)
# [1, 2, 7, 9]
```

University of Exeter

# Objects and Methods

Like functions some methods take input arguments:

*object.method(arg)*  *object.method(arg1, arg2)*

e.g the `.append()` and `.insert()` methods we used to modify lists.

```
x = ['a', 'b']
x.append('c')
print(x)
# ['a', 'b', 'c']
```

```
y = ['a', 'b']
y.insert(1,'X')
print(y)
# ['a', 'X', 'b']
```

University of Exeter

# Objects and Methods

Like functions some methods take input arguments:

*`object.method(arg)`* *`object.method(arg1, arg2)`*

e.g the `.append()` and `.insert()` methods we used to modify lists.

```
x = ['a', 'b']
x.append('c')
print(x)
# ['a', 'b', 'c']
```

```
y = ['a', 'b']
y.insert(1,'X')
print(y)
# ['a', 'X', 'b']
```

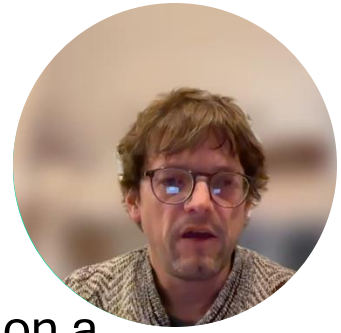
University of Exeter

# Some more useful string methods

| Method | What It Does | Example |
|---|---|---|
| `.replace(old, new)` | Replaces all occurrences of an old substring with a new one. | `'Hi Ann'.replace(Ann', 'Betty')`<br>`→ 'Hi Betty'` |
| `.find(substring)` | Returns the starting index number of the first substring. If not found, it returns -1. | `ada lovelace'.find('love')`<br>`→ 4` |
| `.split(separator)` | Splits the string into a list of smaller strings. If no separator is given, it splits on whitespace. | `'12,1,93'.split(',')`<br>`→ ['12', '1', '93']` |

University of Exeter

# Modules



In addition to its built-in tools, Python allows us to import modules which contain additional tools for particular jobs.

The math library contains the sorts of functions you can access on a scientific calculator.

To use tools from a module we need to load it using the `import` command.


*method #1*

```
from math import sqrt

x = sqrt(16)

print(x)

# 4
```

*loads the* `sqrt` *function from the* `math` *module*

*import individual items from a module*

University of Exeter

# Modules

We can import a whole module to access all its tools.

*method #2*

```
import math
math.exp(1.5)
# 4.48168907
math.pi
# 3.14159265
math.sin(2)
# 0.90929742
```

*here we access items via module name*

*method #3*

```
from math import *
exp(1.5)
# 4.48168907
pi
# 3.14159265
sin(2)
# 0.90929742
```

*here we can directly use items but it's less clear how things belong to the module*

University of Exeter

# More example modules: `os`, `requests`, `numpy`, `pandas`

Dr Philip Lewis

University of Exeter

# The `os` module

The commands used to work with files and folders on different operating systems are different.

*windows*

*linux*

*macOS*

```
> dir
```

*windows command to list directory*

```
$ ls
```

*linux / macOS command to list directory*

The `os` module provides a set of standard commands that will work across different computers.

University of Exeter

# The os module

| Function | What It Does | Example |
|---|---|---|
| `os.getcwd()` | Get Current Working Directory (displays the folder you are in) | `os.getcwd()` → `'C:\\Users\\YourName\\Desktop'` |
| `os.listdir()` | Returns a list of all files and folders in the current directory. | `os.listdir()` → `['mycode.py', 'notes.txt']` |
| `os.mkdir( ____ )` | Make a new directory (folder) with the given name. | `os.mkdir('images')` |

University of Exeter

# The `requests` module

A module that allows us to download data from the web

| Function | What It Does | Example |
|---|---|---|
| `requests.get()` | Sends a GET request to a URL to retrieve data from a server. | `import requests`<br><br>`url = 'https://_____'`<br>`response = requests.get(url)` |

University of Exeter

# The `requests` module

A module that allows us to download data from the web

```
import requests

# The Project Gutenberg site store out-of-copyright books.
# Below we download a book using the requests library

# Web address for "Alice's Adventures in Wonderland"
url = 'https://www.gutenberg.org/files/11/11-0.txt'

response = requests.get(url)

# response.text gives us the full content of the file
full_text = response.text

# display the length of the text in characters
print(len(full_text))

# display the first 300 characters
print(full_text[:250])
```

University of Exeter

# The numpy module

A special module for doing maths on numerical arrays

*It is the convention when working with numpy to rename the* **numpy** *to* **np** *for more compact code*

```
import numpy as np

# Create an example list
mylist = [1, 4, 9, 16]

# Convert the list into a NumPy array
numpy_array = np.array(mylist)

# multiplication on the whole array
numpy_array*2
# Output: [ 2  8 18 32]

# apply numpy function across elements
np.sqrt(numpy_array)
# Output: [1. 2. 3. 4.]
```

University of Exeter

# The numpy module

A special module for doing maths on numerical arrays

```
import numpy as np

# Example creating an array filled with zeros
#
myarray = np.zeros(5)
myarray

# array([0. 0. 0. 0. 0.])
```

University of Exeter

# The numpy module

A special module for doing maths on numerical arrays

```
import numpy as np

# Example an array covering a range
#
#                  start end step
myarray = np.arange( 0,   1, 0.2)
myarray

# array([0.0, 0.2, 0.4, 0.6, 0.8])
```

University of Exeter

# The numpy module

A special module for doing maths on numerical arrays

```
import numpy as np

# Example simulate dice rolls

# np.random.randint() generates
# random ints between two limits (upper bound not included)

#                   lower   upper   size
#.                  bound   bound
np.random.randint( 1,      7,      5.  )

# array([3, 5, 2, 3, 1])
```

*If we are generating random values we will get a different set of values each time we run the code!*

University of Exeter

# The numpy module

A special module for doing maths on numerical arrays

```
import numpy as np

# Example generate values from
# normal (Gaussian) distribution

#                    mean  standard  size
#.                        deviation
np.random.normal( 100,    5,        4)

# array([100.89,  97.96, 104.72 ,  98.39])
```

University of Exeter

# The `pandas` module

- `numpy` allows us to work with arrays of data (tables of numbers)
- `pandas` is a library to work with general data tables (tables of columns containing numbers or text)
- It has similar functionality to Excel in MS Office and the specialised coding language **R** used for statistical analysis.

University of Exeter

# The pandas module

In particular **pandas** makes it very easy to work with files that store data tables in **csv** format (comma separated values)

*Data Table*

| Name | Age | City |
|---|---|---|
| Alice | 25 | Exeter |
| Bob | 30 | London |
| Charlie | 23 | Glasgow |

*csv*

```
Name,Age,City
Alice,25,Exeter
Bob,30,London
Charlie,23,Glasgow
```

*e.g. stored as file:* `people.csv`

University of Exeter

# The pandas module

In particular **pandas** makes it very easy to work with files that store data tables in **csv** format (comma separated values)

*Data Table*

| Name | Age | City |
|---|---|---|
| Alice | 25 | Exeter |
| Bob | 30 | London |
| Charlie | 23 | Glasgow |

*csv*

```
Name,Age,City
Alice,25,Exeter
Bob,30,London
Charlie,23,Glasgow
```

*e.g. stored as file:* `people.csv`

University of Exeter

# The pandas module

```
import pandas as pd

df = pd.read_csv('people.csv')
df
```

*It is the convention when working with numpy to rename the* `pandas` *to* `pd` *for more compact code*

*for large datasets we can use the .head() method to show just the first few rows, e.g.* `df.head()`

| | name | age | city |
|---|---|---|---|
| **0** | Alice | 25 | Exeter |
| **1** | Bob | 30 | London |
| **2** | Charlie | 23 | Glasgow |

```
type(df)
```

```
pandas.core.frame.DataFrame
```

*data is loaded as a* ***DataFrame*** *object class*

University of Exeter

# The pandas module

```
df = pd.read_csv('people.csv')
# extract the age column
# and find the number of entries
len( df['age'] )
```

```
3
```

*we use square brackets to select a column using its name as an index*

```
# find the max age
df['age'].max()
```

```
30
```

```
# find the mean age
df['age'].mean()
```

```
26.0
```

*pandas has built in methods for statistics*

```
# find the standard deviation
# of the age values
df['age'].std()
```

```
3.605551275463989
```

University of Exeter

# Installing modules

- Some modules (e.g. `math`, `os` ) are included as standard.
- If you install Python as part of Anaconda (or Jupyterlab Desktop) then additional modules for data analysis will already be installed and ready to use in notebooks (e.g. `numpy`, `pandas` )
- In Jupyterlab you can install modules easily using `pip` (package installer for Python). You can run this in a code cell.

*error if a module is not installed*

```
import doesnotexist
```

```
---------------------------------------
ModuleNotFoundError
Cell In[58], line 1
----> 1 import doesnotexist
```

*how to install a module with pip*

```
pip install numpy
```

```
Requirement already satisfied: numpy
```

University of Exeter

# Summary of example modules

- **`numpy`** and `pandas` allow us to work with data using special **`array`** and **`DataFrame`** objects.
- **`pandas`** makes it easy to read in data from csv formatted files.
- The **`requests`** module is also useful for accessing data from the web.
- The **`os`** module is useful for simple file / folder operations.

# Making Plots with Matplotlib

Dr Philip Lewis

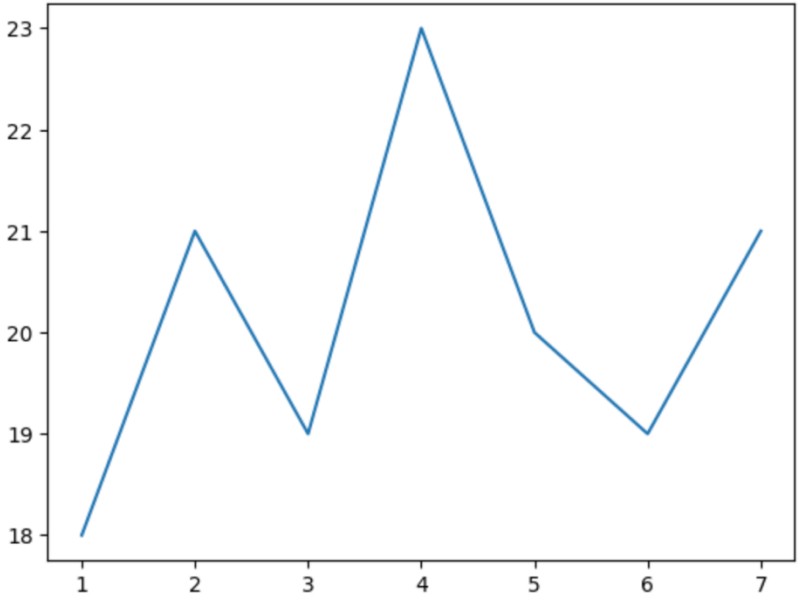
University of Exeter

# Plotting data

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

```
x_vals = [  1,  2,  3,  4, 5,   6,  7 ]
y_vals = [ 18, 21, 19, 23, 20, 19, 21 ]

plt.plot(x_vals, y_vals)
```

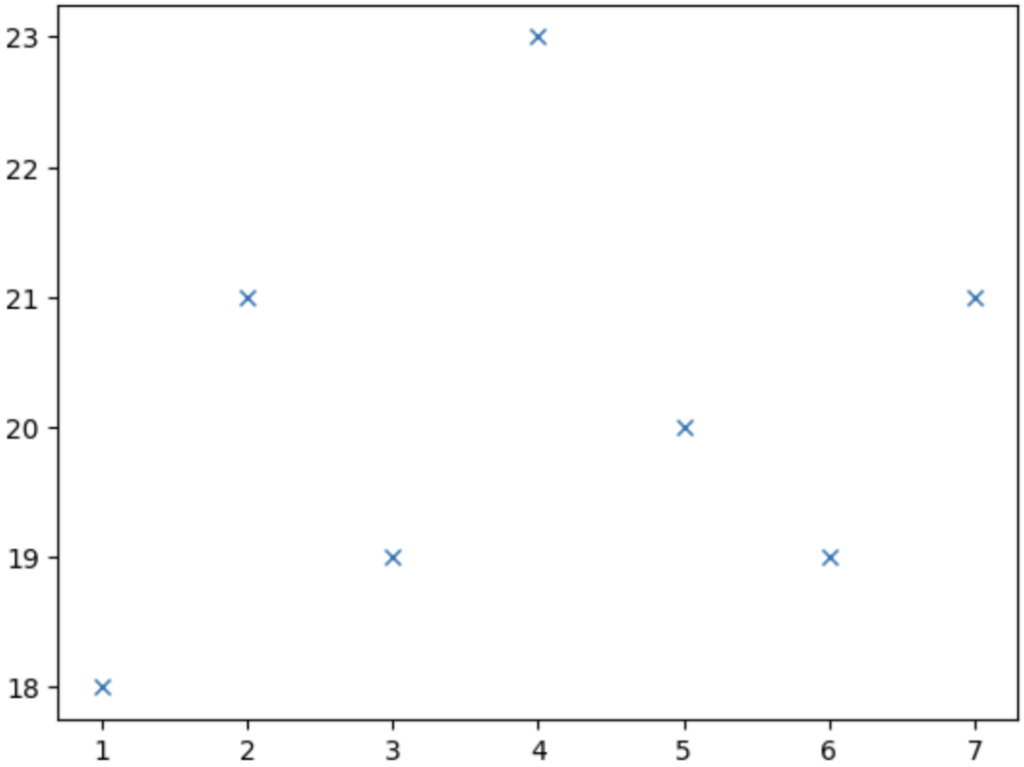



University of Exeter

# Plotting with marker only

```
plt.plot(x_vals, y_vals, 'x')
```

```
[<matplotlib.lines.Line2D at 0x1202fb470>]
```




```
plt.plot(x_vals, y_vals, 'x')
plt.plot(x_vals, y_vals, 'o')
plt.plot(x_vals, y_vals, '^')
plt.plot(x_vals, y_vals, '.')
```

University of Exeter

# Quick plot style:

```
plt.plot(x_vals, y_vals, 'bx-')
```

quick style

**color**

```
b blue      g green     r red      c cyan
m magenta   y yellow    k black    w white
```

**marker**

```
. dots      * star
o circles   ^ triangles
x cross     + alt. cross
```

**line**

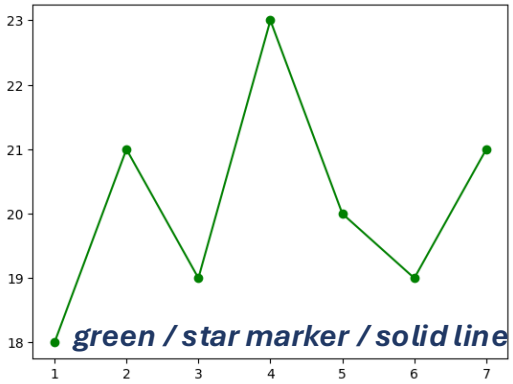
```
- solid line      : dotted line   -- dashed line
```

University of Exeter

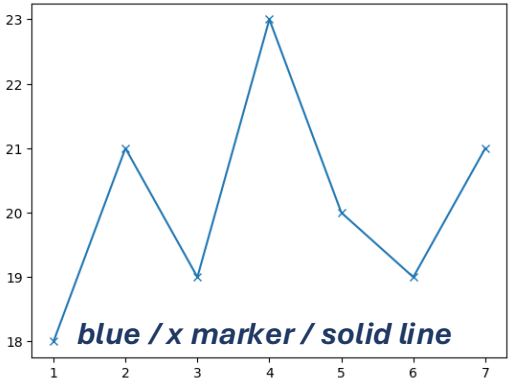
# Quick plot style: color / marker / line

```
plt.plot(x_vals, y_vals, 'r+-')
```

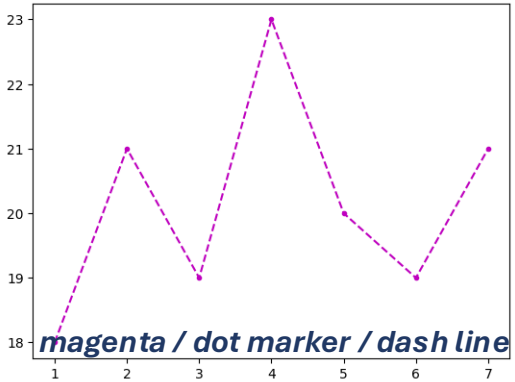

```
plt.plot(x_vals, y_vals, 'go-')
```



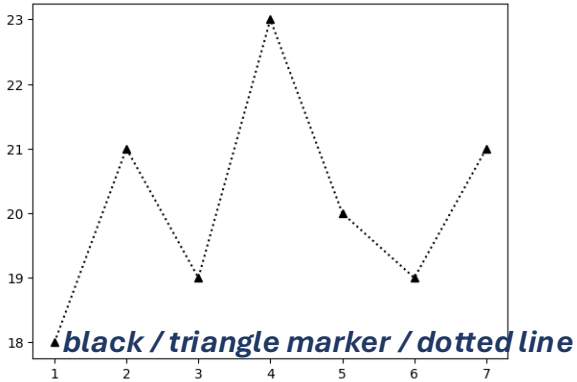

```
plt.plot(x_vals, y_vals, 'bx-')
```




```
plt.plot(x_vals, y_vals, 'm.--')
```



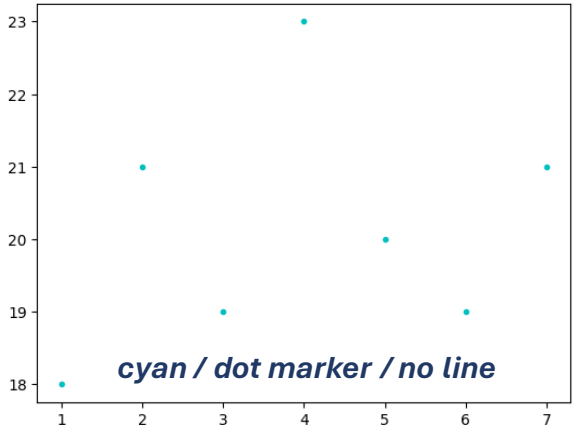

```
plt.plot(x_vals, y_vals, 'k^:')
```



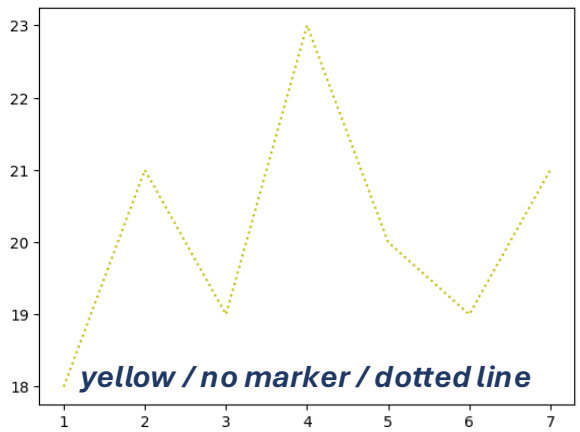

University of Exeter

# Quick plot style: color / marker / line

```
plt.plot(x_vals, y_vals, 'c.')
```




```
plt.plot(x_vals, y_vals, 'y:')
```




alternative way to specify styles in full

```
plt.plot(x_vals, y_vals, linestyle='dotted', color='yellow')
```

**see w3 tutorials:**
**https://www.w3schools.com/python/matplotlib_intro.asp**

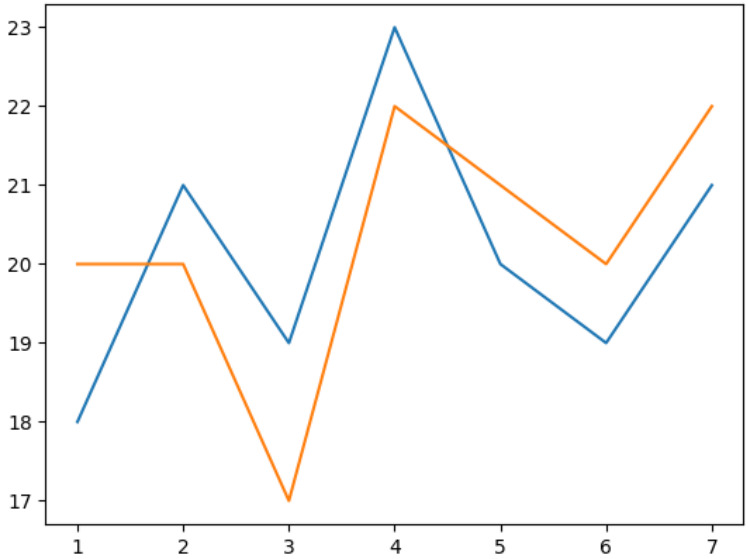
University of Exeter

# Multiple trendlines

```
# multiple data series

x_vals = [  1,  2,  3,  4, 5,   6,  7 ]
y1_vals = [ 18, 21, 19, 23, 20, 19, 21 ]
y2_vals = [ 20, 20, 17, 22, 21, 20, 22 ]

plt.plot(x_vals, y1_vals)
plt.plot(x_vals, y2_vals)
```



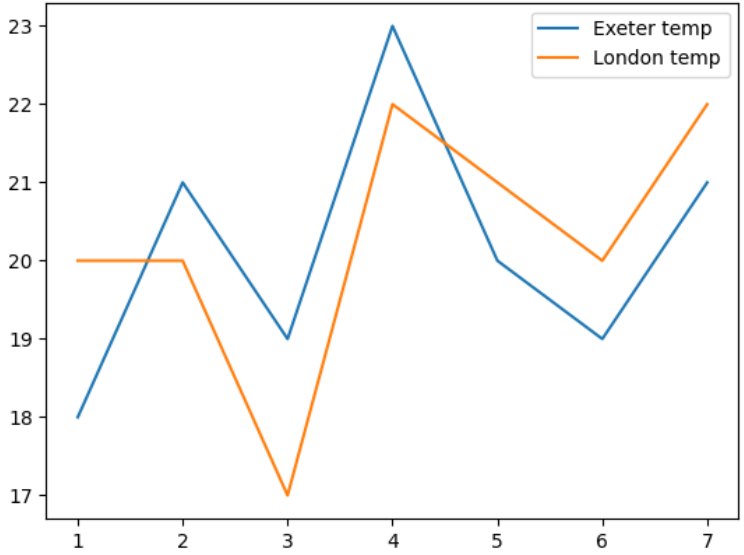

University of Exeter

# Multiple trendlines

```
# multiple data series

x_vals = [  1,  2,  3,  4, 5,   6,  7 ]
y1_vals = [ 18, 21, 19, 23, 20, 19, 21 ]
y2_vals = [ 20, 20, 17, 22, 21, 20, 22 ]

plt.plot(x_vals, y1_vals, label='Exeter temp')
plt.plot(x_vals, y2_vals, label='London temp')
plt.legend()
```




University of Exeter

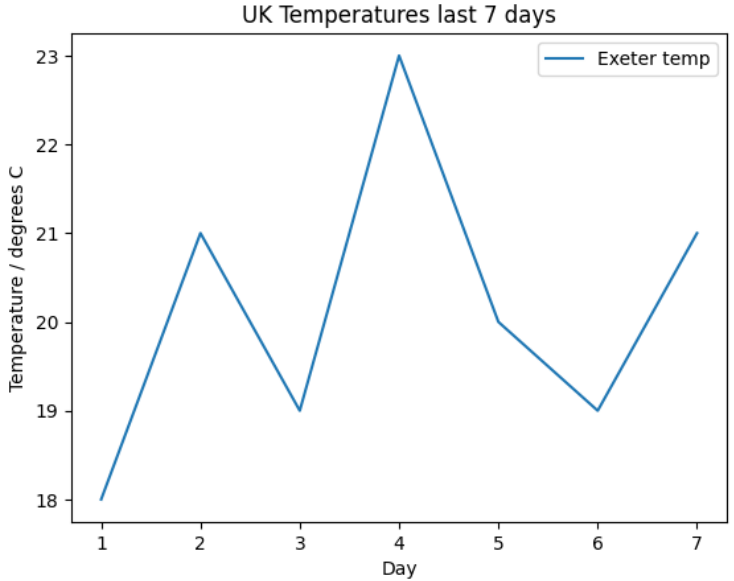
# Title and axes labels

```
# title and axes labels

x_vals = [  1,  2,  3,  4, 5,   6,  7 ]
y_vals = [ 18, 21, 19, 23, 20, 19, 21 ]

plt.plot(x_vals, y_vals, label='Exeter temp')
plt.legend()

plt.title('UK Temperatures last 7 days')
plt.xlabel('Day')
plt.ylabel('Temperature / degrees C')
```




University of Exeter

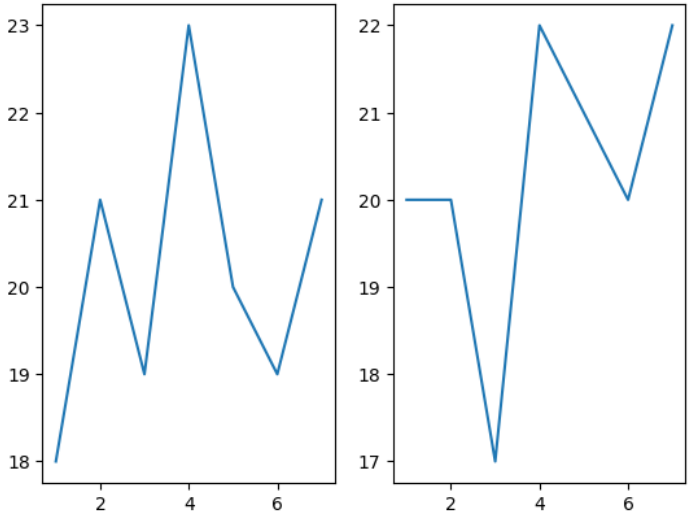
# Multiple plots using `subplot`

```
x_vals = [  1,  2,  3,  4, 5,   6,  7 ]
y1_vals = [ 18, 21, 19, 23, 20, 19, 21 ]
y2_vals = [ 20, 20, 17, 22, 21, 20, 22 ]

plt.subplot(1, 2, 1)
#the figure has 1 row, 2 columns, working in subplot 1
plt.plot(x_vals, y1_vals)

plt.subplot(1, 2, 2)
#the figure has 1 row, 2 columns, working in subplot 2
plt.plot(x_vals, y2_vals)
```

1 row
2 columns




University of Exeter

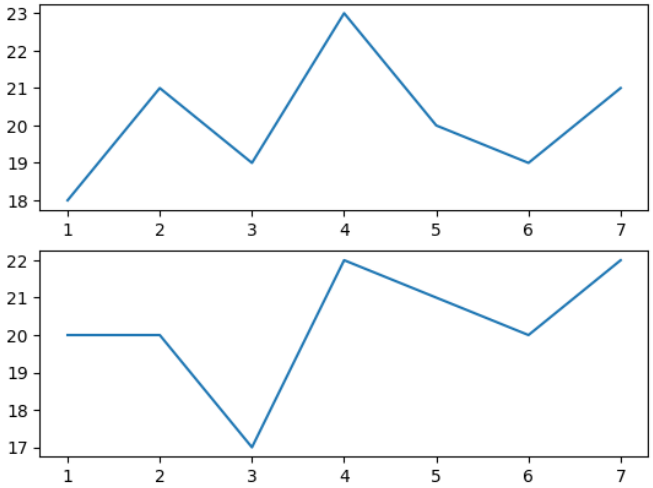
# Multiple plots using `subplot`

```
x_vals = [  1,  2,  3,  4, 5,   6,  7 ]
y1_vals = [ 18, 21, 19, 23, 20, 19, 21 ]
y2_vals = [ 20, 20, 17, 22, 21, 20, 22 ]

plt.subplot(2, 1, 1)
#the figure has 2 rows, 1 column, working in subplot 1
plt.plot(x_vals, y1_vals)

plt.subplot(2, 1, 2)
#the figure has 2 rows, 1 column, working in subplot 2
plt.plot(x_vals, y2_vals)
```

2 rows
1 column




University of Exeter

# Labelling multiple plots

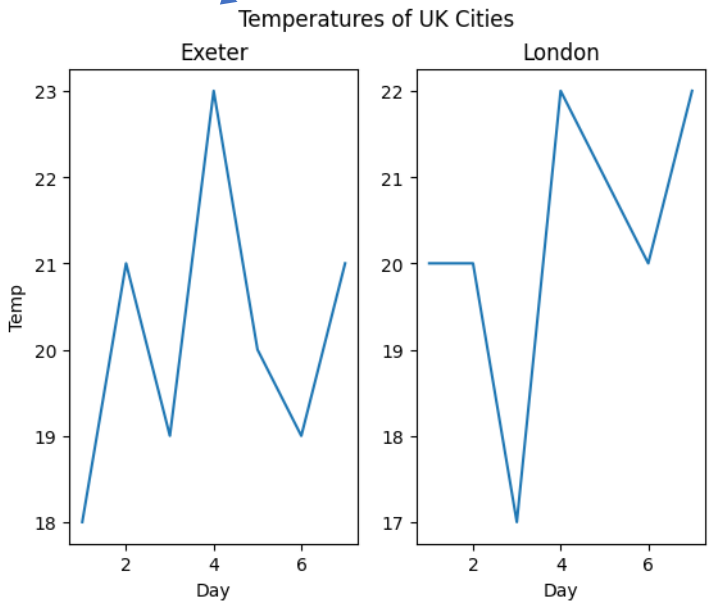
*super title*

```
x_vals = [  1,  2,  3,  4, 5,   6,  7 ]
y1_vals = [ 18, 21, 19, 23, 20, 19, 21 ]
y2_vals = [ 20, 20, 17, 22, 21, 20, 22 ]

plt.subplot(1, 2, 1)
#the figure has 2 rows, 1 column, working in subplot 1
plt.plot(x_vals, y1_vals)
plt.title('Exeter')
plt.ylabel('Temp')
plt.xlabel('Day')


plt.subplot(1, 2, 2)
#the figure has 2 rows, 1 column, working in subplot 2
plt.plot(x_vals, y2_vals)
plt.title('London')
plt.xlabel('Day')

plt.suptitle('Temperatures of UK Cities')
```




University of Exeter

# Customising Plots with Matplotlib

Dr Philip Lewis

University of Exeter

# Customising

You can configure all plot aspects!

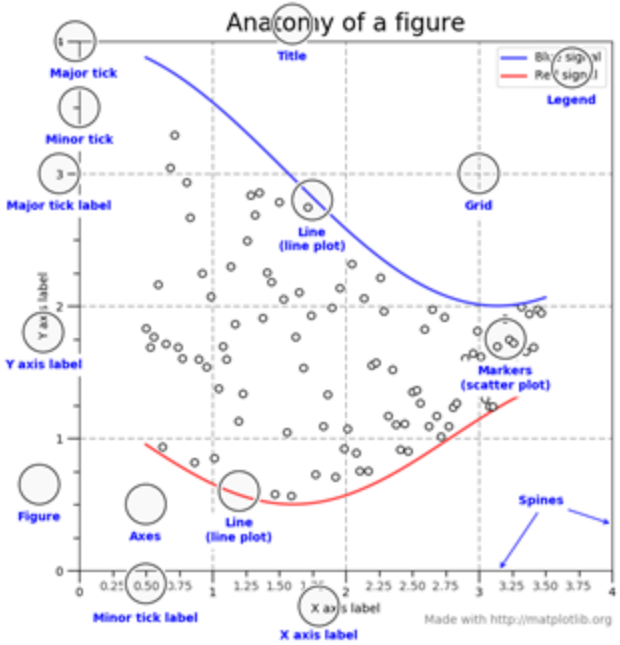
*link to tutorials: https://www.w3schools.com/python/matplotlib_intro.asp*

schools Tutorials Refer

HTML CSS JAVASCRIPT

Python Matplotlib

- Matplotlib Intro
- Matplotlib Get Started
- Matplotlib Pyplot
- Matplotlib Plotting
- Matplotlib Markers
- Matplotlib Line
- Matplotlib Labels
- Matplotlib Grid
- Matplotlib Subplot




Rather than try to learn everything, get the basics and ask Google / GenAI what you want to do in a particular case.

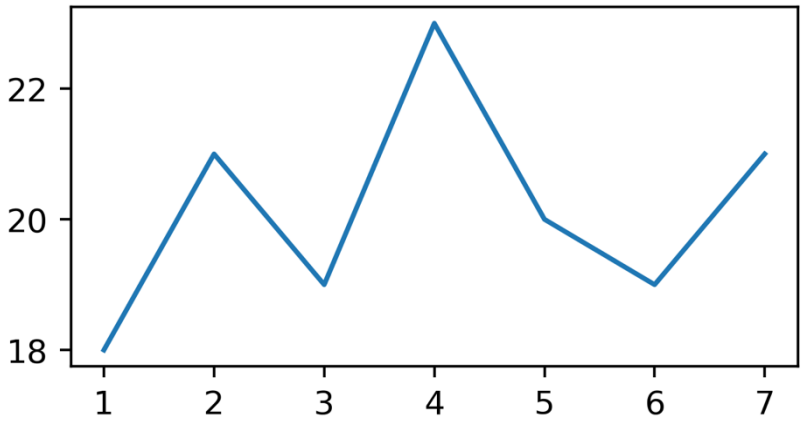
University of Exeter

# Customising plots

You can configure all plot aspects!

*link to tutorials: https://www.w3schools.com/python/matplotlib_intro.asp*

```
plt.figure(figsize=(4, 2), dpi=400)
plt.plot(x_vals, y_vals)
```




*To set a custom figure width, height we create the figure before we plot.*

*You can also improve the sharpness of the image by setting the dpi (dots per inch)*

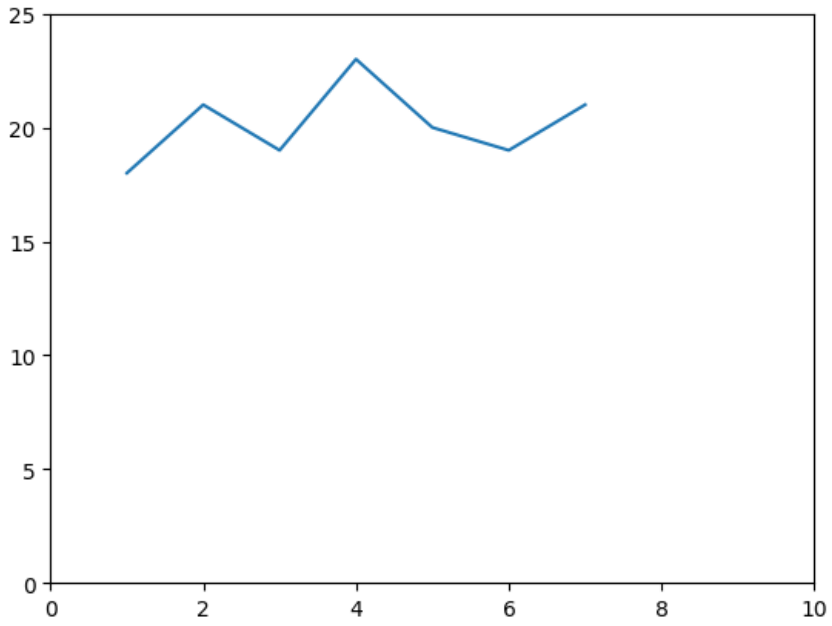
University of Exeter

# Customising axes limits

You can configure all plot aspects!

*link to tutorials: https://www.w3schools.com/python/matplotlib_intro.asp*

```
plt.plot(x_vals, y_vals)
plt.ylim(0,25)
plt.xlim(0,10)
```




*matplotlib sets the axes range to fit the plotted data, but often we might need to adjust this*

University of Exeter

# Customising axes limits

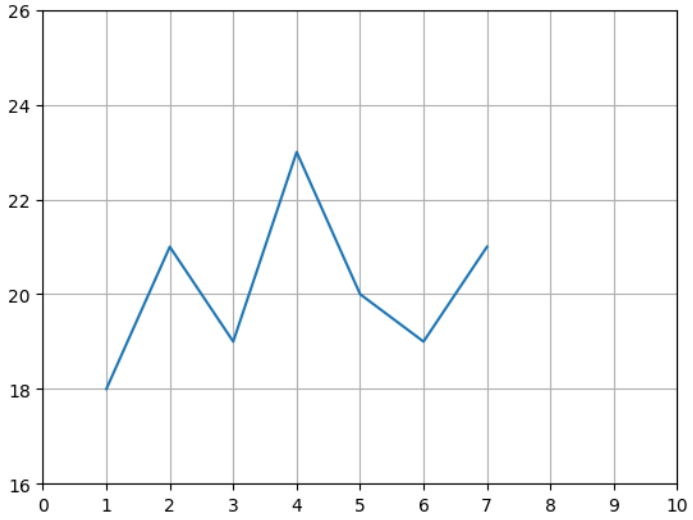
You can configure all plot aspects!

*link to tutorials: https://www.w3schools.com/python/matplotlib_intro.asp*

```
import numpy as np

plt.plot(x_vals, y_vals)
plt.xticks(np.arange(0,10.1,1))
plt.yticks(np.arange(16,27,2))

plt.grid()
```




*we can specify tick intervals (e.g. using numpy's arange function)*
*and draw on a grid for clarity*

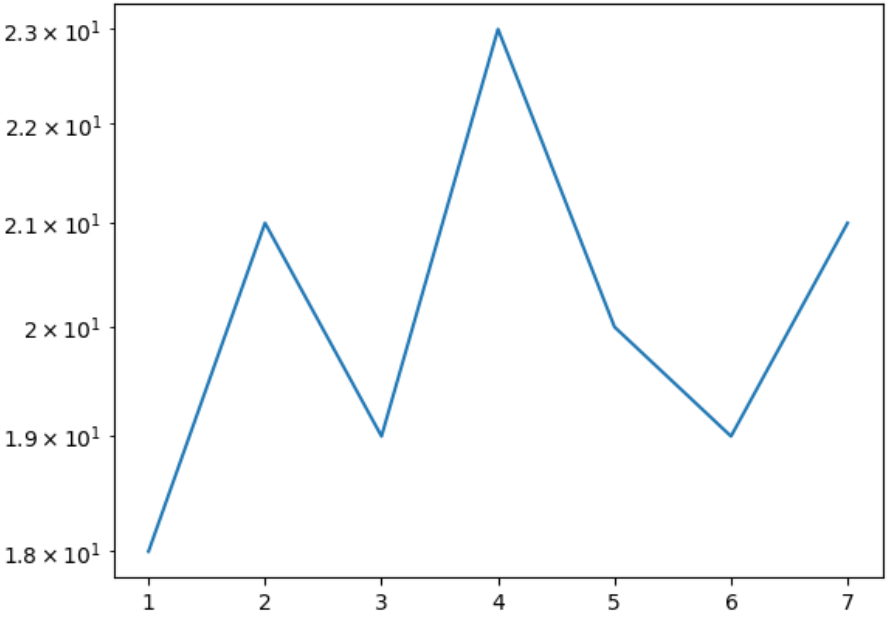
University of Exeter

# Customising axes limits

You can configure all plot aspects!

*link to tutorials: https://www.w3schools.com/python/matplotlib_intro.asp*

```
plt.plot(x_vals, y_vals)
plt.yscale('log')
```




*where data spans a wide range, using log axes can help us visualise the data from the smallest and largest scale*

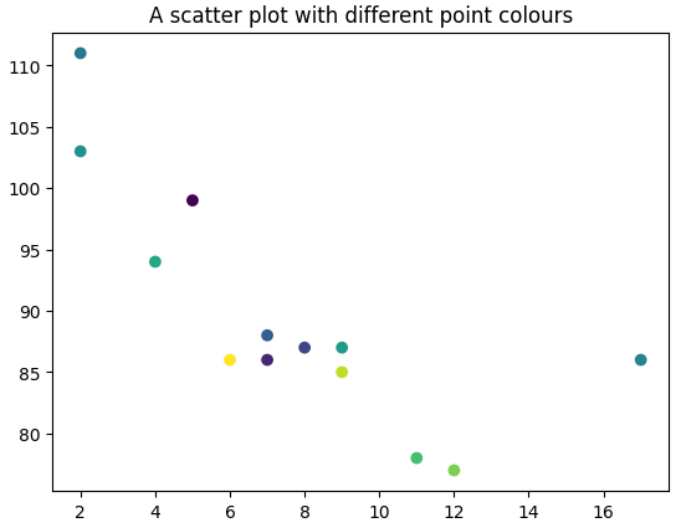
University of Exeter

# Scatter plots: point colours

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

x_vals = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y_vals = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
n_vals = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])

plt.scatter(x, y, c=n_vals, cmap='viridis')
plt.colorbar
plt.title('A scatter plot with different point colours')
```




*using plt.scatter() to make a scatter plot lets us customise points further.*

*e.g. where we have additional data on each point we can plot an extra variable using a colour mapping to translate values to different colours.*

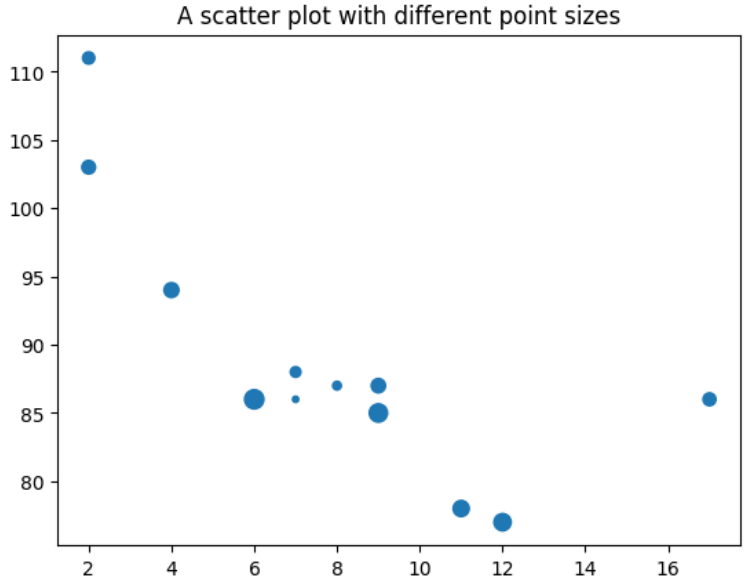
University of Exeter

# Scatter plots: point sizes

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

x_vals = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y_vals = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
n_vals = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])

plt.scatter(x, y, s=n_vals)
plt.title('A scatter plot with different point sizes')
```




*using plt.scatter() to make a scatter plot lets us customise points further.*

*e.g. we can use the additional data to set the size of each point*

University of Exeter

# Plot types in Matplotlib

Dr Philip Lewis

University of Exeter

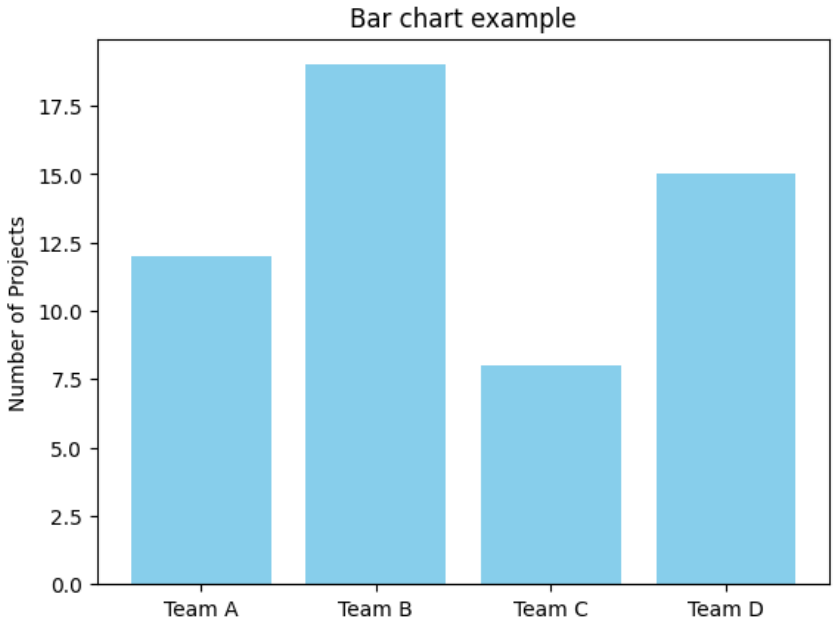
# Bar chart

```
import matplotlib.pyplot as plt

teams = ['Team A', 'Team B', 'Team C', 'Team D']
projects_completed = [12, 19, 8, 15]

plt.bar(teams, projects_completed, color='skyblue')

plt.title('Bar chart example')
plt.ylabel('Number of Projects')
```

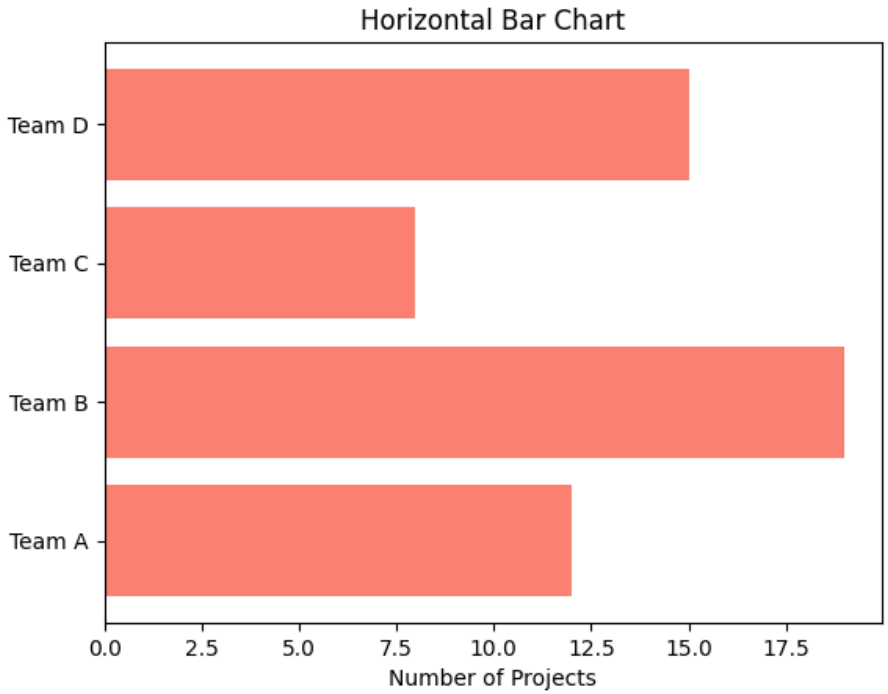



University of Exeter

# Horiztonal bar chart

```
plt.barh(teams, projects_completed, color='salmon')

plt.title('Horizontal Bar Chart')
plt.xlabel('Number of Projects')
```



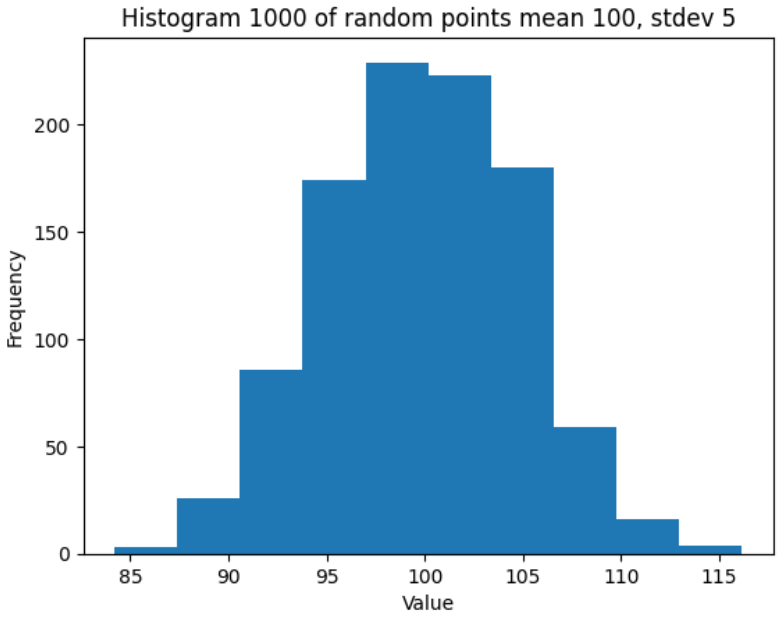

University of Exeter

# Histogram

```
import numpy as np
data_points = np.random.normal(100, 5, 1000)
```

```
plt.hist(data_points)

plt.title('Histogram 1000 of random points mean 100, stdev 5')
plt.xlabel('Value')
plt.ylabel('Frequency')
```

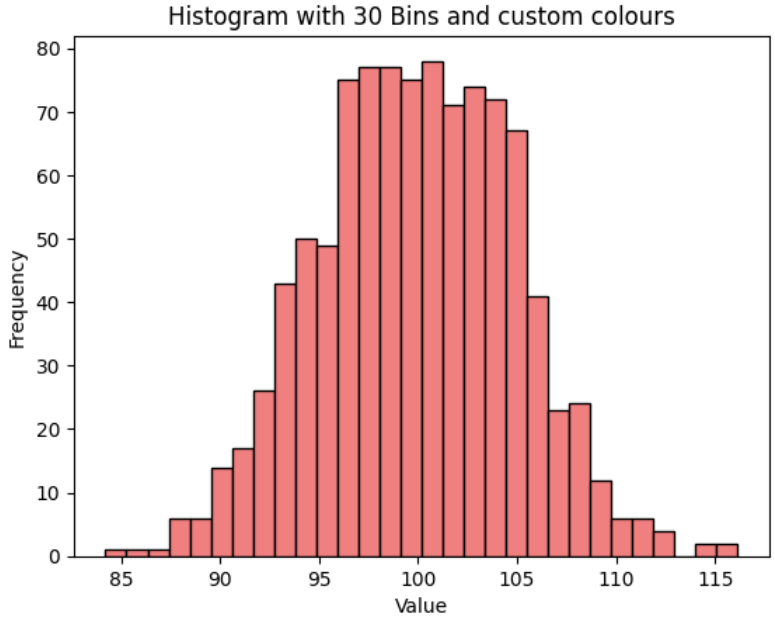



University of Exeter

# Histogram: custom number of bins

```
plt.hist(data_points, bins=30, color='lightcoral', edgecolor='black')

plt.title('Histogram with 30 Bins and custom colours')
plt.xlabel('Value')
plt.ylabel('Frequency')
```

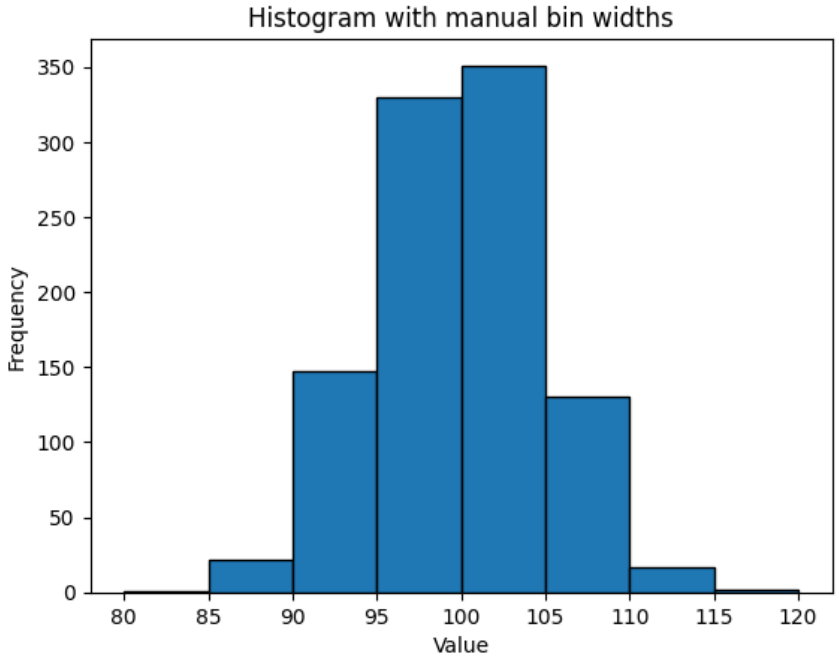



University of Exeter

# Histogram: custom bin values

```
plt.hist(data_points, bins=np.arange(80, 121, 5), edgecolor='black')

plt.title('Histogram with manual bin widths')
plt.xlabel('Value')
plt.ylabel('Frequency')
```



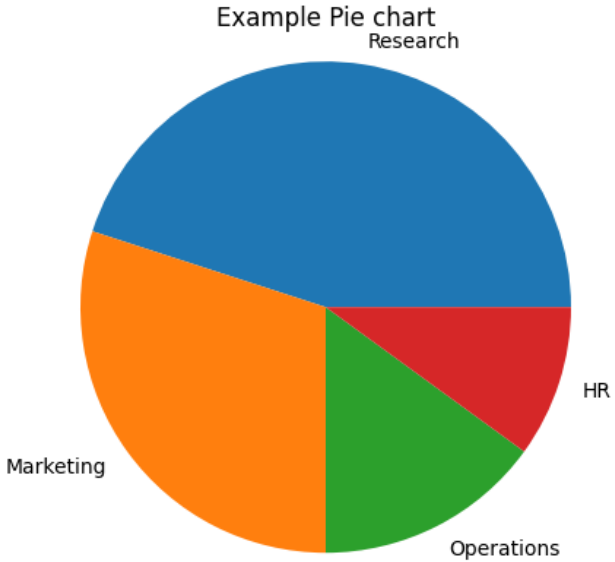

University of Exeter

# Pie chart

```
labels = ['Research', 'Marketing', 'Operations', 'HR']
sizes = [45, 30, 15, 10]  # Percentages

plt.pie(sizes, labels=labels )

plt.title('Example Pie chart')
plt.axis('equal')  # Ensures the pie chart is a circle.
```



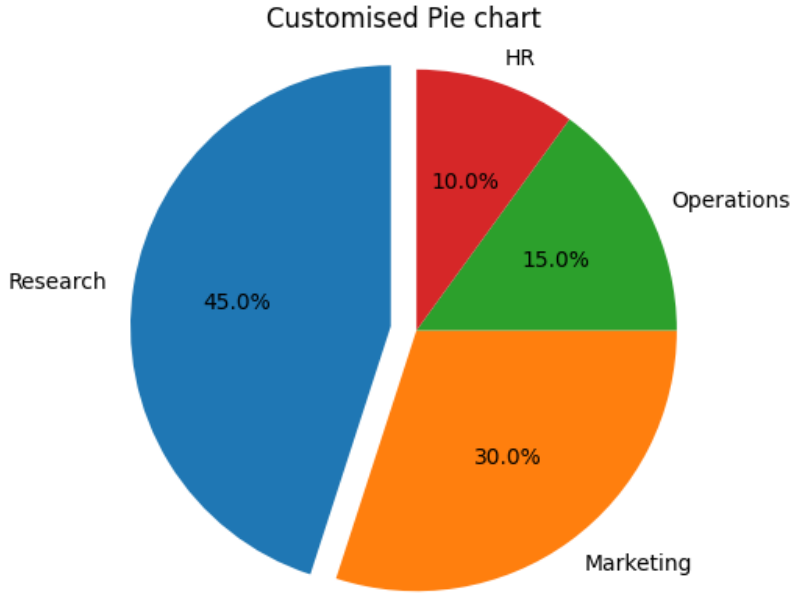

University of Exeter

# Customised Pie chart

```
labels = ['Research', 'Marketing', 'Operations', 'HR']
sizes = [45, 30, 15, 10]  # Percentages
explode = (0.1, 0, 0, 0)  # "Explode" the 1st slice (Research)

plt.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%', startangle=90)

plt.title('Customised Pie chart')
plt.axis('equal')  # Ensures the pie chart is a circle.
```



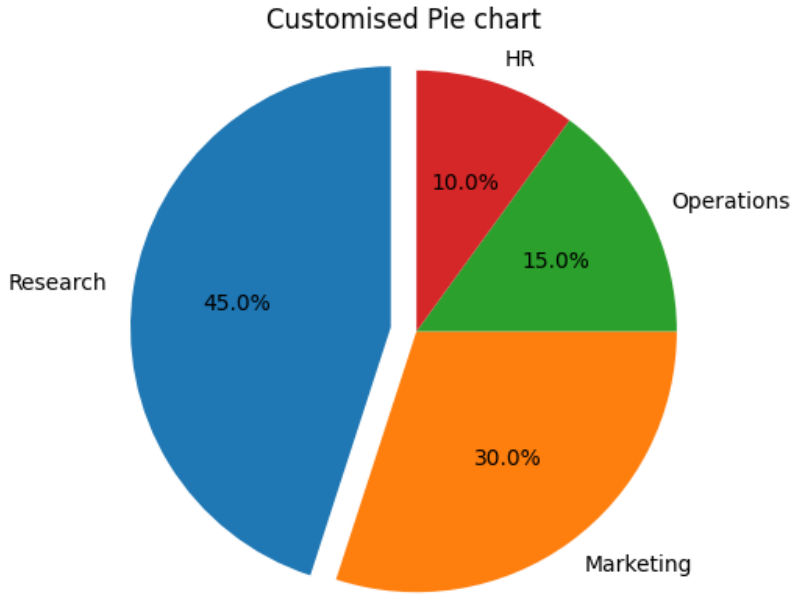

University of Exeter

# Customised Pie chart

```
labels = ['Research', 'Marketing', 'Operations', 'HR']
sizes = [45, 30, 15, 10]  # Percentages
explode = (0.1, 0, 0, 0)  # "Explode" the 1st slice (Research)

plt.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%', startangle=90)

plt.title('Customised Pie chart')
plt.axis('equal')  # Ensures the pie chart is a circle.
```




University of Exeter