



Institute of
Data

2023



Introductory Python for Data Science



Agenda

- Learning outcomes
- Course contents
- Questions



Learning outcomes

- [Basic skills in programming](#) in Python for data science including:
 - Use Jupyter Notebook to write and run Python code
 - Find and read the [Python documentation](#) for libraries and functions
 - Work with basic [Python data types](#) (string, float, integer, list, etc.)
 - Write [Python expressions](#) that involve variables, variable assignment, operators and functions
 - Use [Python conditional and loop](#) functions
 - Resolve coding errors
 - Create basic graphs
 - [Read, clean and manage data](#)



Course contents

- Overview
- **Programming** as a **problem-solving** technique
 - Lab 1
- Python **environments, tools and key libraries**
 - Lab 2
- **Data analysis** in Python and data analysis projects
 - Lab 3
- Summary and call for action

Questions



Introductory Python for Data Science

Module 1

Part 1

Programming as problem-solving



Agenda

- What is programming?
- Importance of programming for data science
- Python Fundamentals
- Developing and running Python
- Input and output function options
- Data structures in Python
- Writing functions in Python
- Iterating in Python



What is programming?

- Programming is:
 - the process of creating a set of **instructions** that tell a computer how to perform a **task**.
 - thinking **systematically** and **critically**
 - breaking a task into steps. Examples include: a **recipe**, **directions** to a destination and **mathematical** problem solving
- A program usually takes an **input** and produce an **output**
- You can think of programming as a way to **solve a problem** to generate the **required output** from a **given input**
- Difference between **programming** and **coding**?
 - Programming is the skill to specify a program **independent** of any programming language
 - Coding is writing the program in a specific **programming language**



Programming is a fundamental skill for data science

- Data science involves **problem solving** at many levels and in each step of a project in an implicit (conceptual) or explicit form (programs)
- **Programming**, which is the main tool for data science, can be defined in its essential form as a problem-solving technique for data-driven problems
- **Python** is the most popular programming language for data science

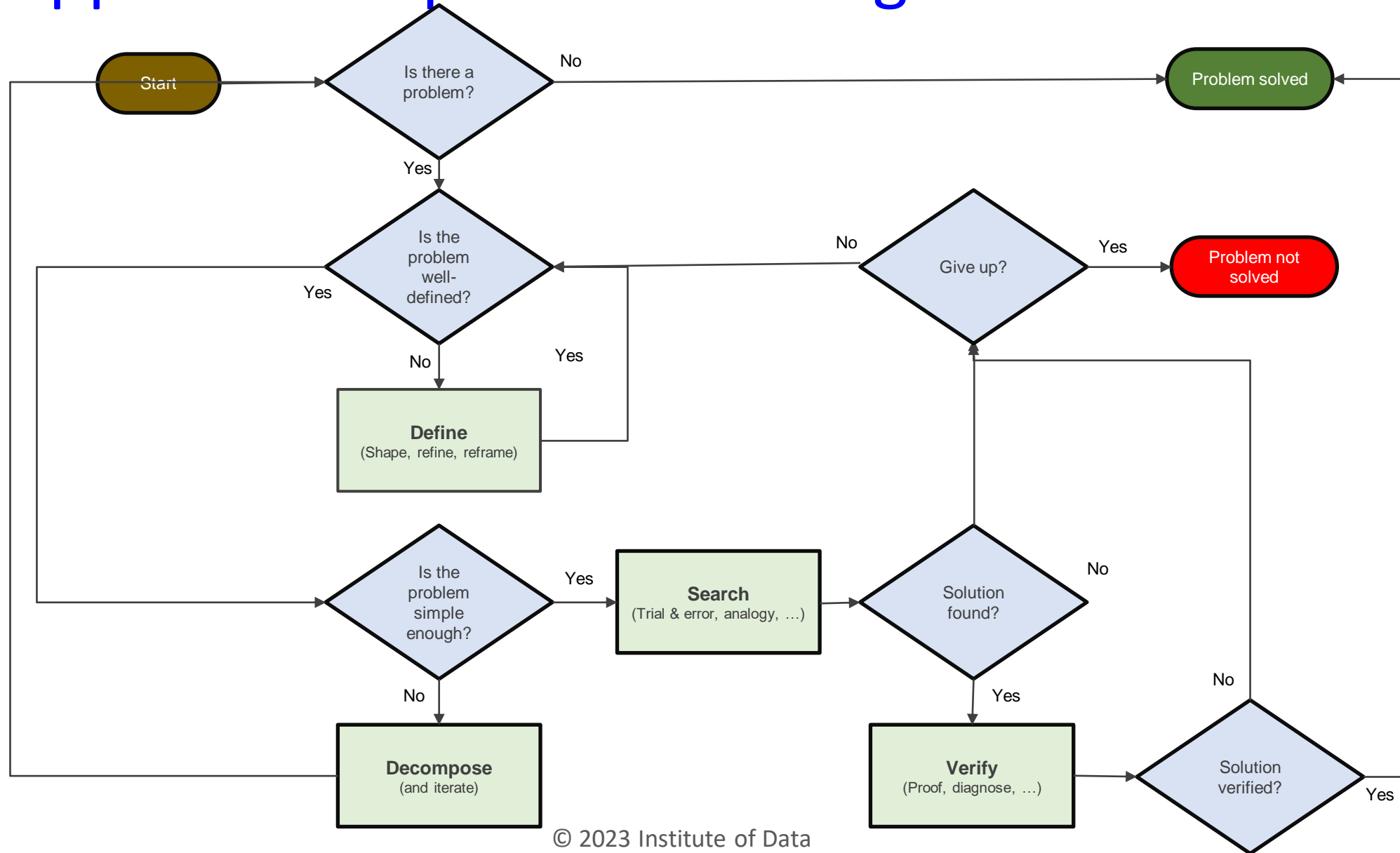


Problem solving

- Problem solving is the *process* of finding solutions to difficult or complex issue
- Scientific method involves stating problems in a manner that facilitate solving them *mathematically* and verify them *empirically*
- Problems can be solved using techniques that include a combination of the following actions:
 - Define
 - Decompose
 - Search
 - Validate



An approach for problem solving





Python programming language

- Python is a [high-level programming language](#), and its core design philosophy is all about code [readability](#) and a syntax which allows programmers to express concepts in a few lines of code
- Python is used for developing many different types of computer programs including:
 - [Data analysis](#)
 - [Data visualisation](#)
 - [Machine learning](#)



Python comments

- Single line comment

```
# This is a comment
```

- Multiple line comments

```
"""
```

```
Multiple line comment
```

```
"""
```



Python input and output functions

- Input function receives an input from the user

```
Data = input('Please enter your name:')
```

- Print function prints formatted text and variables

```
A = 100
```

```
Print(f"This is a text and embedded variable {A}")
```



Python variables and data types

- **Variables** are used to store **information** to be referenced and manipulated in a computer program.
- Common data types
 - **Integer** <int> examples: 1, 1095, -2
 - **Float** <float> examples: 1.2, - 2974.074
 - **String** <str> examples: 'Bob', "This is a longer string \t with special char's"
 - **Boolean** <bool> examples: True, False
- Python allows you to **convert** variables between these types when needed
- **Type** command
 - `type(12.65) -> <class 'float'>`



Python operations

- Math operations
 - + plus - minus / divide * multiply
 - < less-than > greater-than <= less-than-equal
 - >= greater-than-equal
- Logic operations
 - and, or, not



If/else, for loops, while loops

- The **if/else** statement executes a block of code if a **specified condition** is true. If condition is not met, another block of code can be executed.
- **Loops** through a block of code a **number of times**
- **Loops** through a block of code while a specified **condition** is met
- continue
- break
- pass

```
var = 10
if (var >= 5):
    print('var is greater than or equal 5')
elif (var < 0):
    print('var is negative')
elif (var == 0):
    print('var is zero')
else:
    print('var is less than 5')
```

```
for i in range(10):
    print(i)
```

```
var = 10
while(var < 20):
    print('var is less than 20')
    var+=2
```



Data structures

- **lists**

- A list is the Python equivalent of an array, but is resizable and can contain elements of different types
- Functions: append, extend, insert, remove, pop, clear, index, count, sort, reverse, copy
- comprehensions

- **tuples**

- A tuple is an (immutable) ordered list of values.

- **sets**

- A set is an unordered collection with no duplicate elements.

- **dictionaries**

- A dictionary stores (key, value) pairs

```
Tuple_x = (2, 7)
```

```
List_y = [2, 4, 6, 8]
```

```
Dictionary_z = {"id": 123,  
"name": "Item 123"}
```



Functions

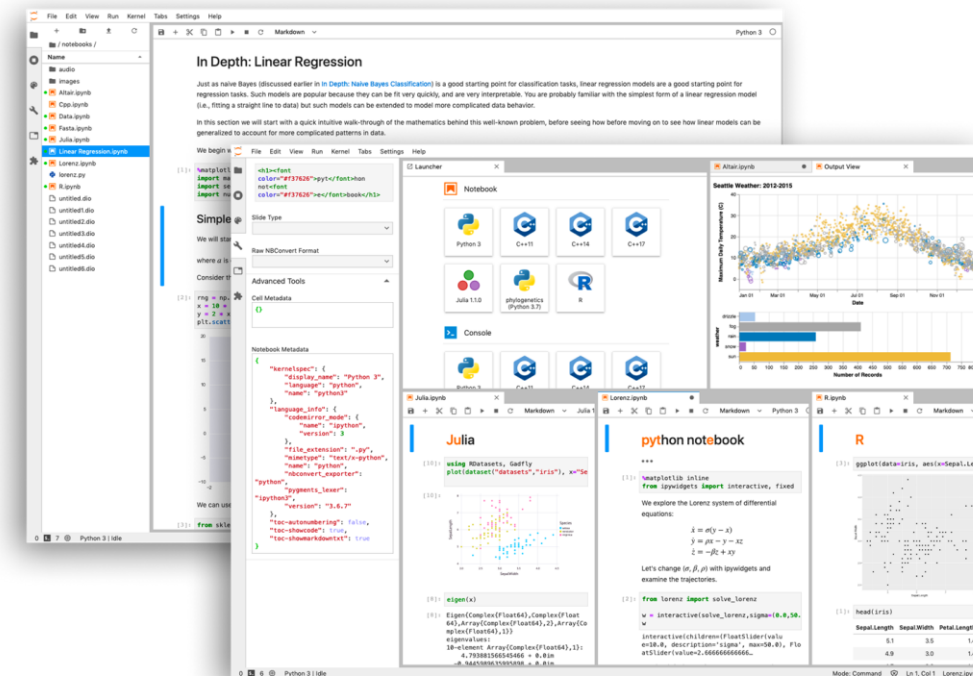
```
def funcName(param1, param2, defArg1 = 0, defArg2 = 100):  
    # code here  
    return someResult
```

- **Optional parameters** take default arguments if missing from function call
- **Arguments** are assigned to parameters in defined sequence unless named in call
- **return** statement
 - optional
 - can return multiple items
- **scope** is inherited from main (but not from a calling function)



Jupyter notebook

- The **Jupyter** notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualisations and narrative text
- We will use Jupyter notebooks for exercises in this course





Lab 1: First Python program

- Read and follow the instructions in the notebook



Introductory Python for Data Science

Part 2

Python environment, tools and libraries



Agenda

- Python environment
- Python tools
- Python libraries



Environments

What is an environment?

A practical way to deal with Python's packages (libraries)

Issues:

- Many packages have not been around long enough to be tested with other packages that you might want to use with them
- Packages don't always get updated quickly in response to updated dependencies

Solution:

- Create virtual environments for hosting isolated projects using [Anaconda](#) Navigator



Installing Packages with pip

- install a package
- upgrade a package
- install a specific version
- install a set of requirements
- install from an alternate index
- install from a local archive

```
$ pip install anypkg
```

```
$ pip install --upgrade anypkg
```

```
$ pip install anypkg==1.0.4
```

```
$ pip install -r reqsfile.txt
```

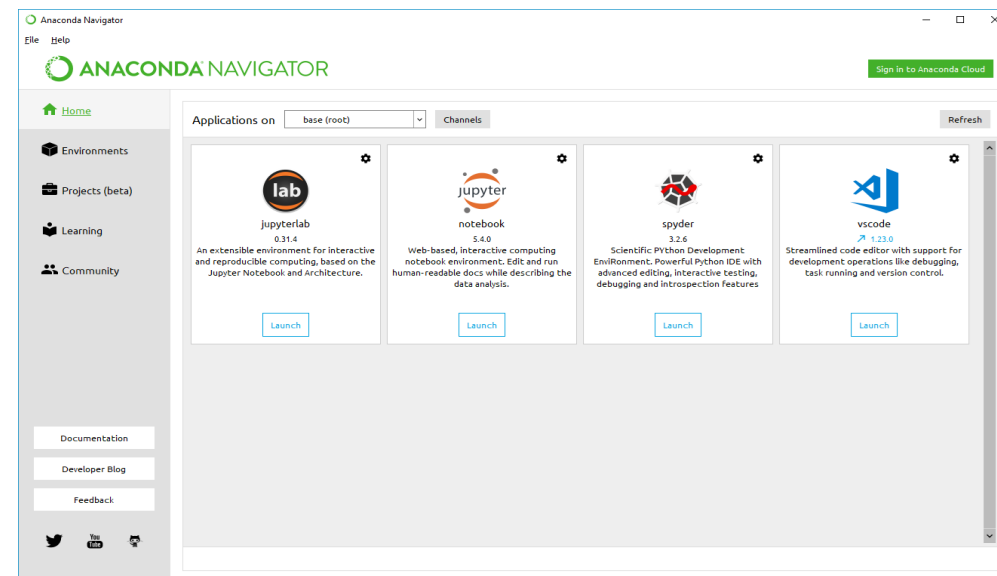
```
$ pip install --index-url  
http://my.package.repo/simple/ anypkg
```

```
$ pip install ./downloads/anypkg-  
1.0.1.tar.gz
```



Anaconda

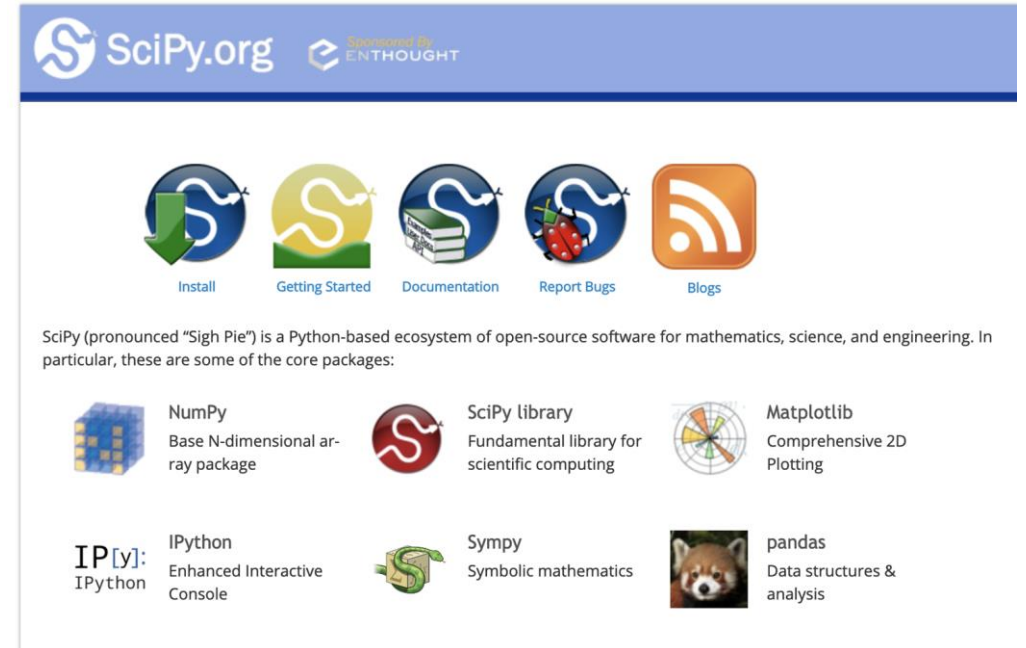
Anaconda Distribution is **the recommended way** to configure and manage your Python development and running environment(s).





SciPy

- SciPy (pronounced “Sigh Pie”) is a Python-based ecosystem of open-source software for mathematics, science, and engineering
- Main libraries (packages) include numpy, scipy, matplotlib, ipython, jupyter, pandas, sympy, nose



<https://www.scipy.org/>



Numpy

- Numpy is the fundamental package for scientific computing with Python
- A powerful N-dimensional array object
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities and many, many more



Numpy data types

Type	Python	Numpy	Usage
byte byte array	<code>b'any string'</code> <code>bytearray()</code>		<ul style="list-style-type: none">• immutable• mutable
integer	<code>int()</code>	<ul style="list-style-type: none">• 11 types	<ul style="list-style-type: none">• signed, unsigned• 8, 16, 32, 64 bits, unlimited
floating-point	<code>float()</code>	<ul style="list-style-type: none">• 3 types	<ul style="list-style-type: none">• 16, 32, 64 bits
complex	<code>complex()</code>	<ul style="list-style-type: none">• 2 types	<ul style="list-style-type: none">• 64, 128 bits
unassigned	<code>None</code>		<ul style="list-style-type: none">• object• <code>myVar is not None</code>
missing	<code>nan</code>	<code>isnull()</code> , <code>notnull()</code> , <code>isnan()</code>	<ul style="list-style-type: none">• float, object



Visualisation libraries

matplotlib

- histograms
- bars
- curves
- surfaces
- contours
- maps
- legends
- annotations
- primitives

Seaborn

- based on matplotlib
- prettier
- more informative
- more specialised



Lab 2: Python libraries

- Read and follow the instructions in the notebook

Questions



Introductory Python for Data Science

Part 3

Data analysis in Python



Agenda

- Introduction to **data analysis**
- Data sources and shapes
- Data analysis operations
- **Pandas library**
- Data **loading**
- Data **visualisation**
- Data **statistics**
- Data **insights**



Data sources and shapes

- Where does data come from?
 - [Databases](#)
 - Transaction systems
 - [Websites](#)
- What data looks like?
 - Database tables
 - [Spreadsheets](#)
 - Structured or semi-structured files



Data analysis operations

- **Wrangling**
 - Sourcing, loading, and precleaning the data so we can see what it really looks like
- **Profiling**
 - Visualising and understanding the essential characteristics of the data
- **Munging**
 - reshaping the data to prepare it for analysis



Pandas library

- Rich relational data analysis tool built on top of NumPy
- Easy to use and highly performing APIs
- A foundation for data wrangling, munging, preparation, etc in Python

	Name	Passport	Flight
0	John Muir	Z1248227	EK424
1	Ansel Adams	Z1248229	EK525
2	James Savage	Z1248242	LY126
3	Galen Clark	Z1248269	6E025

Pandas Data Frame



Loading and exploring data

- Pandas can load data from many sources including csv files, websites and databases
- Pandas load data into a data structure called a Data Frame which looks like a spreadsheet

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
weather =
pd.read_csv('https://raw.githubusercontent.com/
alanjones2/dataviz/master/london2018.csv')
print(weather.head())
```

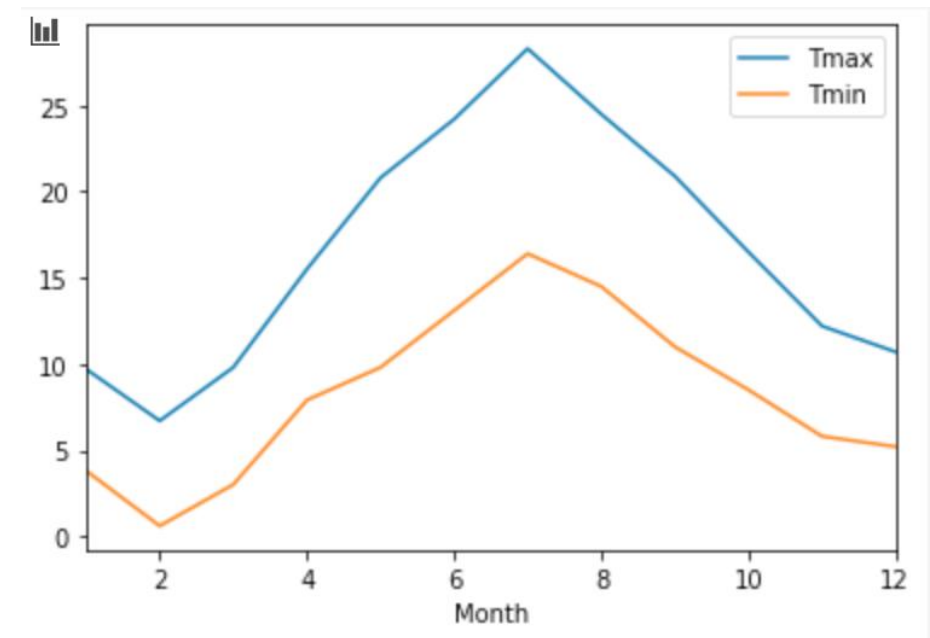
	Year	Month	Tmax	Tmin	Rain	Sun
0	2018	1	9.7	3.8	58.0	46.5
1	2018	2	6.7	0.6	29.0	92.0
2	2018	3	9.8	3.0	81.2	70.3
3	2018	4	15.5	7.9	65.2	113.4
4	2018	5	20.8	9.8	58.4	248.3



Data visualisation

- Data can be plotted directly from pandas' data frames using matplotlib
- There are many plot types available including:
 - Line chart
 - Bar chart
 - Scatter plot
 - Pie chart
 - Histograms
 - etc

```
weather.plot(y=['Tmax', 'Tmin'], x='Month')
```





Data statistics

- Pandas provides many functions that allow you to explore statistics of the data including:
 - Count
 - Mean
 - Standard deviation
 - Minimum
 - Maximum

```
from sklearn.datasets import load_iris
dataset=load_iris()
data=pd.DataFrame(dataset["data"],columns=["Petal length","Petal Width","Sepal Length","Sepal Width"])
data["Species"]=dataset["target"]
data["Species"]=data["Species"].apply(lambda x: dataset["target_names"][x])
print(data.head())
print(data.describe())
```

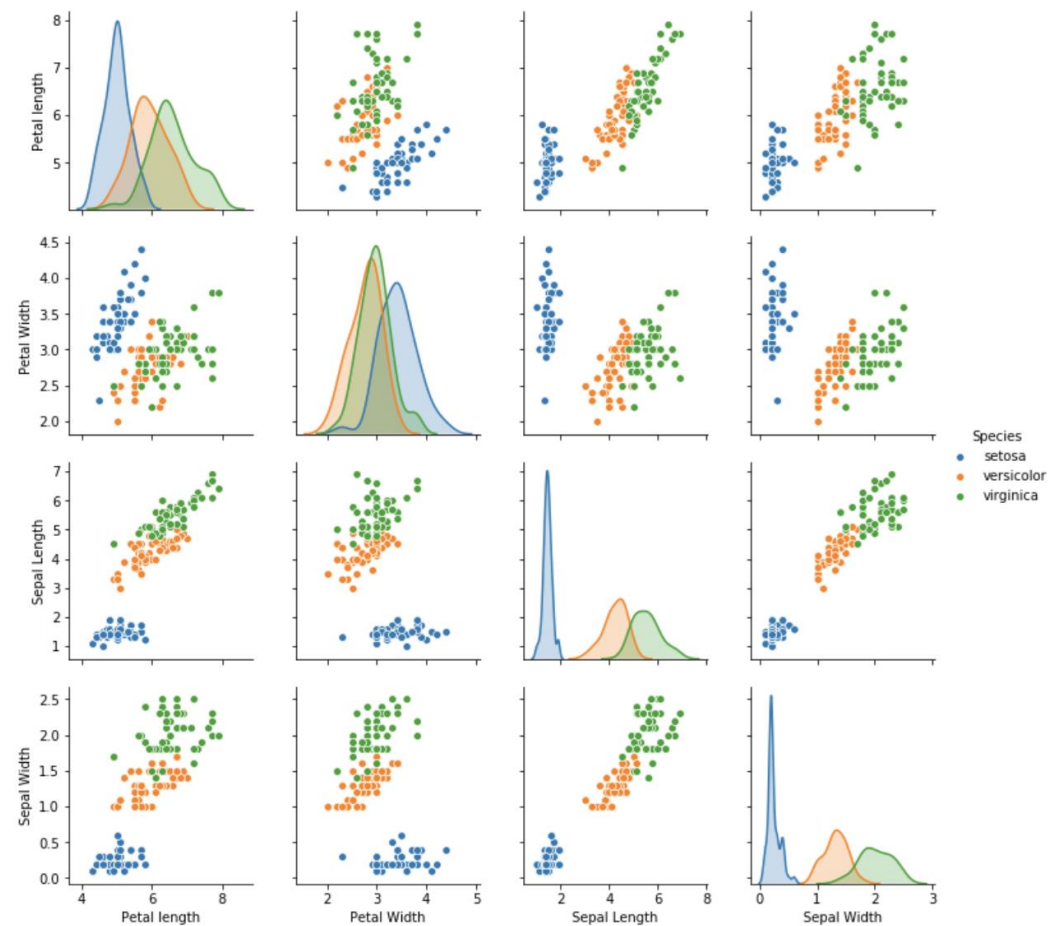
	Petal length	Petal Width	Sepal Length	Sepal Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
	Petal length	Petal Width	Sepal Length	Sepal Width	
count	150.000000	150.000000	150.000000	150.000000	
mean	5.843333	3.057333	3.758000	1.199333	
std	0.828066	0.435866	1.765298	0.762238	
min	4.300000	2.000000	1.000000	0.100000	
25%	5.100000	2.800000	1.600000	0.300000	
50%	5.800000	3.000000	4.350000	1.300000	
75%	6.400000	3.300000	5.100000	1.800000	
max	7.900000	4.400000	6.900000	2.500000	



Analytical insights

- Using pandas, numpy and matplotlib you can not just describe and visualise the data. You can obtain insights that show deeper relationships between various data elements.

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





Lab 3: Data analysis in Python

- Read and follow the instructions in the notebook

Questions



Introductory Python for Data Science

Part 4

Summary and call for action



Summary and call for action

- We explored what is **programming** and how it can be viewed as a **problem-solving** technique.
- We introduced **Python** as a suitable programming language for implementing data science projects
- We applied programming and data analysis techniques in a number of lab **exercises** that hopefully gave you a flavour of how data analysts, scientists and engineers use Python to perform **data-driven** projects.

Questions

End of presentation