**Week 0 – Data visualisation with Python**

**Introduction**

This series of exercises is designed to focus on helping you take your first steps in data visualisation with python using a practical approach. We will only cover basic programming techniques, so please use in conjunction with other resources, and python documentation.

Heavy use of LLMs such as ChatGPT while very useful are not recommended while learning, as you cannot know if the output is sensible, if you have not learnt the basics of how to use python.

**Why choose python?**

As always your choice of tools should depend on:

- Industry standards in your target field, along with colleague expertise
- Your experience (i.e., the speed of delivering solutions vs. learning new tools)
- Strengths and weaknesses of the toolset; some rough thoughts are below, but this list is neither exhaustive nor detailed, and some will be subjective.

**Strengths**

- Python is one of the most widely used programming languages today, with applications across academia, industry, and research.
- Python provides access to many of the most up-to-date and widely adopted machine learning and data science libraries.
- The language is considered easy to learn and read, due to its clear syntax and emphasis on readability.
- A large ecosystem of third-party packages and libraries is available for most common tasks.
- Python is a general-purpose language and can be used for a wide variety of applications, from scripting to web development and data analysis.
- Python is an interpreted language, meaning no separate compilation step is required.
- Strong community support and extensive online documentation make troubleshooting and learning easier.
- Python enables rapid development, allowing programs to be written and iterated on quickly.
- Python is cross-platform and runs consistently on Windows, macOS, and Linux.
- Python integrates well with other technologies, including C/C++, Java, and web frameworks.

**Weaknesses**

- Python is a general-purpose language and does not excel in every area.

For example, building graphical user interfaces (GUIs) can be more challenging compared to tools designed specifically for asynchronous event-driven systems, where long-running tasks do not block the interface.

- Python's syntax is not part of the C-family of languages.

This can make it harder for developers to transfer knowledge directly between Python and languages such as C, C++, or Java.

- Python provides relatively few features out of the box.

Most real-world tasks rely on external libraries, which must be installed and learned.

- While performance continues to improve, Python has traditionally prioritised readability and accessibility over raw execution speed.
- Heavy reliance on high-level libraries can obscure what operations are occurring "under the hood," making performance characteristics and implementation details less transparent.

**Contents**

There are eight Jupiter Notebooks written for colab. There is no assessment for this course, your assessment will focus on your application of data visualisation theory. This course aims to introduce and get you started with tools you might use to support this.

| | | |
|---|---|---|
| General worksheets | - Week 1: Barcharts with pandas, and matplotlib<br>- Week 2: Scatter plot with plotly<br>- Week 3: Line charts and faceting with plotly | A core selection of tools that will allow you to create a complete data visualisation. |
| Extending the possibilities | - Week 4: Pie charts and conditional logic<br>- Week 5: Heatmaps with Plotly Graph Objects<br>- Week 6: KPI's and text | Some very useful but not necessarily essential examples. |
| Advanced topics | - Week 7: Maps and Geo information<br>- Week 8: Interactive dashboards with Dash | Commonly requested but more advanced tools. |

**Background**

**Libraries**

Python contains relatively few specialised native methods and functions compared to many of its competitors. For example, advanced numerical operations are not built directly into the core language. As a result, many of the techniques we will need are provided through libraries.

Libraries can be roughly divided into two types:

- Those included in the standard library, which are usually available by default.

- External libraries, which must be installed separately.

In Google Colab, many common external libraries are already included in the environment. Once installed, libraries must be imported into a script, conventionally at the top of the file. In a notebook environment, these imports are typically placed in a separate cell. Try to avoid repeatedly running import cells unnecessarily, as some libraries can take time to load.

**NumPy**

NumPy is included in Colab and provides many mathematical and numerical functions for Python. It is largely implemented in C and C++, giving it a reputation for being both fast and reliable. NumPy arrays are significantly faster than Python lists for numerical operations and support vectorised computation, allowing operations to be applied to whole arrays without explicit loops.

**Pandas**

Pandas (also included in Colab) brings R-style dataframes to Python. It relies on NumPy for low-level numerical operations and provides convenient tools for analysing tabular data. While pandas is extremely powerful and easy to use, poorly written operations can lead to performance issues, particularly on large datasets. Many operations that are slow in pandas can be made much faster using NumPy or more specialised tools. Despite this, pandas remains an excellent choice for exploring and working with small to medium-sized datasets, with many of the skills that you will learn using this library being able to be applied to higher performance tools.

**Matplotlib**

Matplotlib (included in Colab) was originally developed to bring MATLAB-style plotting to Python. It renders plots as image outputs (e.g. PNG). It is often considered the most basic yet widely used plotting library. Despite its simplicity from the user's perspective, due to its extensive internal dependency chain, Matplotlib installation can occasionally be non-trivial, especially when using the latest Python releases. For our dashboard, we will only use Matplotlib to get started before moving to plotly, but Seaborn built on top of Matplotlib is a great alternative.

Matplotlib (included in Colab) was originally developed to bring MATLAB-style plotting to Python. It renders plots as image outputs (e.g. PNG). It is often considered the most basic yet widely used plotting library. Despite its simplicity from the user's perspective, due to its extensive internal dependency chain, Matplotlib installation can occasionally be non-trivial, especially when using the latest Python releases. For our dashboard, we will only use Matplotlib to get started before moving to plotly, but Seaborn built on top of Matplotlib is a great alternative.

**Plotly**

Plotly is a JavaScript-based visualisation library built on top of D3.js, and it renders plots using a web browser (or a browser-like viewer). Because of this, Plotly visualisations are interactive by default, supporting features such as zooming, panning, and tooltips. In addition, being built on D3.js makes Plotly highly customisable and well-suited for modern, interactive data exploration.

**Plotly Express**

A version of plotly built on plotly that is quicker and simpler to use.

**Jupiter Notebook**

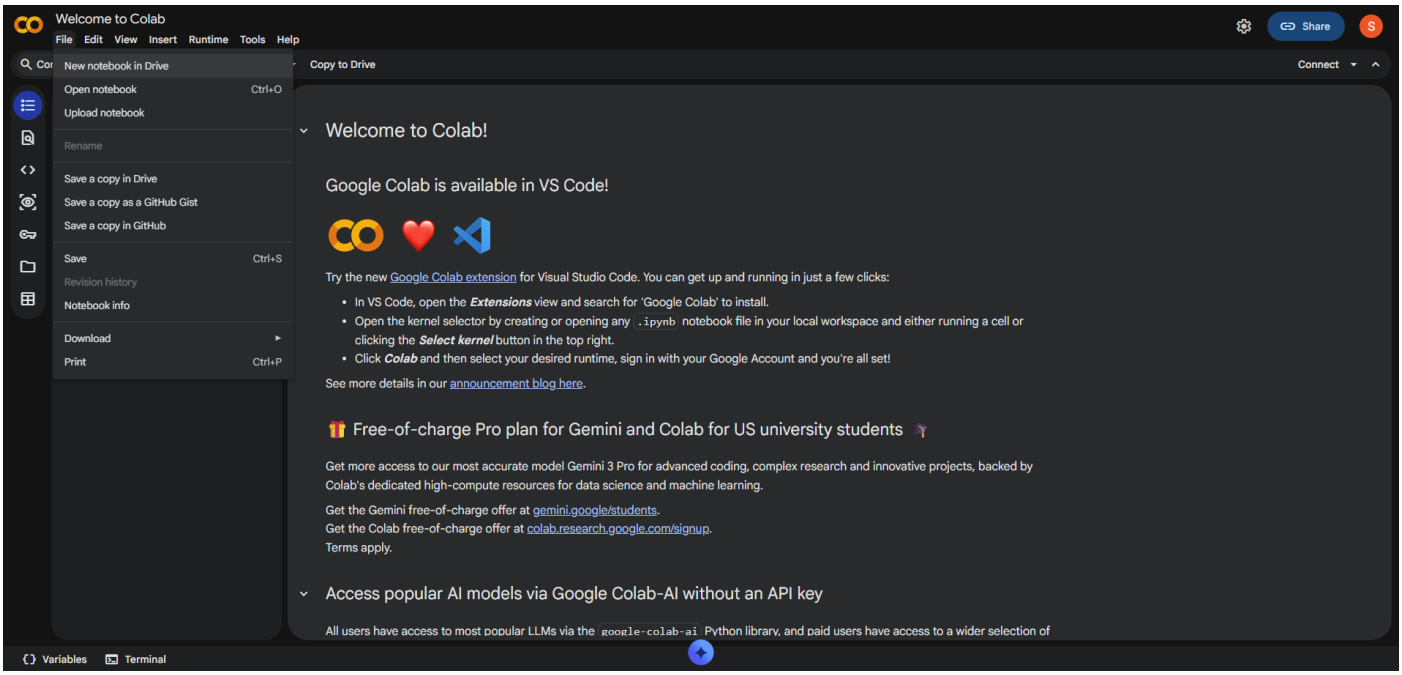
A python script is normally run from a .py file. A python notebook (.pynb) is a type of file that allows for code blocks and markdown text setions.

**Google Colab**

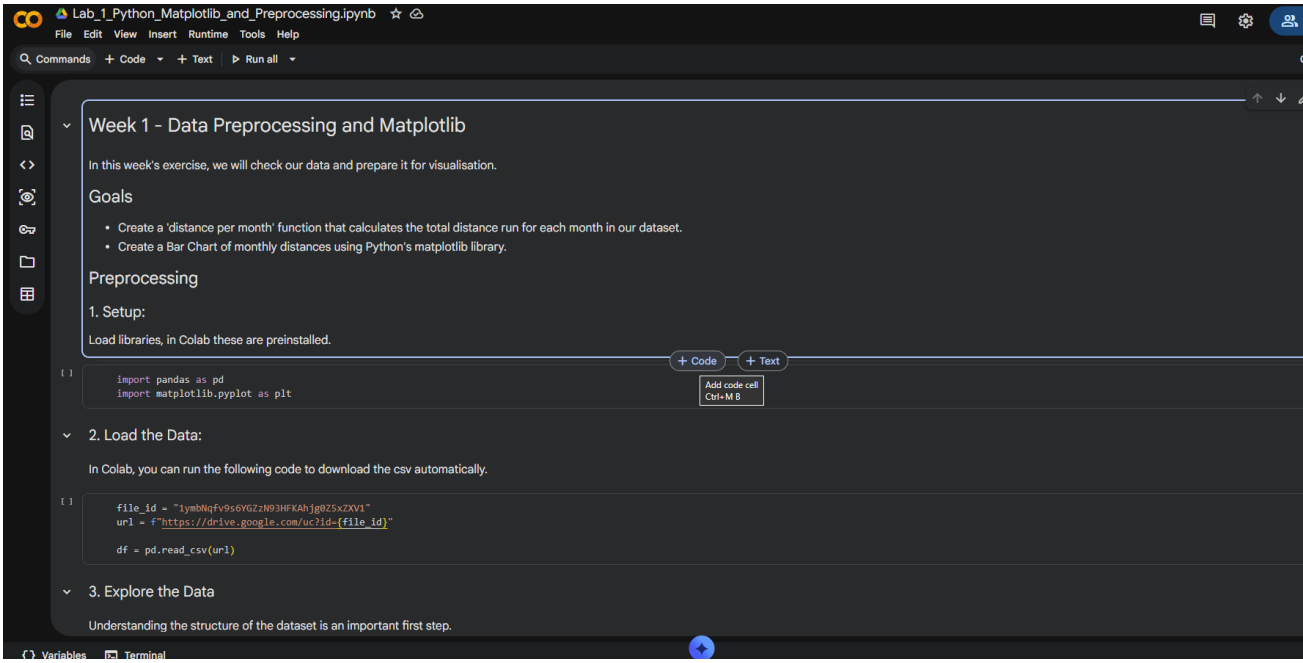
Colab is a notebook editor connected to google drive that allows for code to be run using a pre packaged cloud python environment.

**Getting Started with Google Colab**

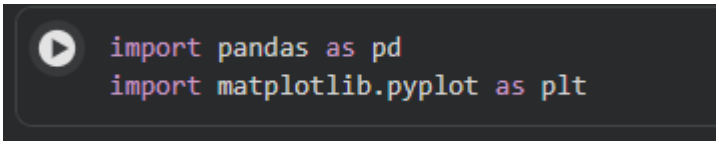
1) Download week 1.pynb from learning central.
2) Navigate to https://colab.research.google.com/, and log in with a required google account.




3) Using File -> Upload, load the notebook into colab.

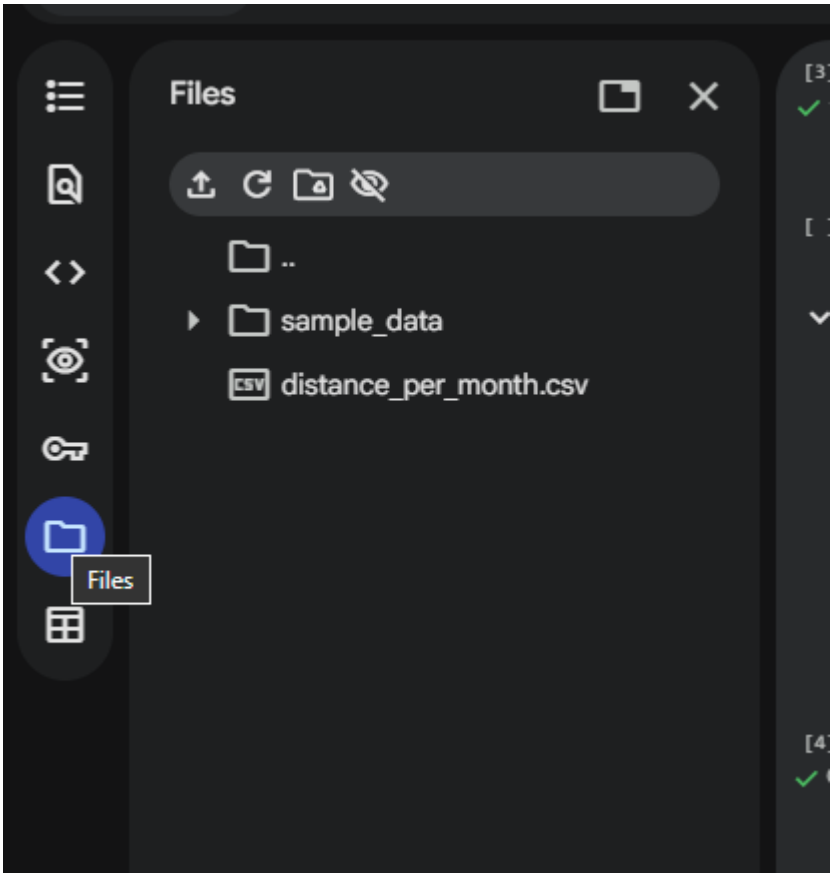



4) Hovering between the sections we can create ‘code blocks’ or ‘text blocks’
5) A code block can be run by pressing the arrow. When it has been any variable created will be usable throughout the notebook.

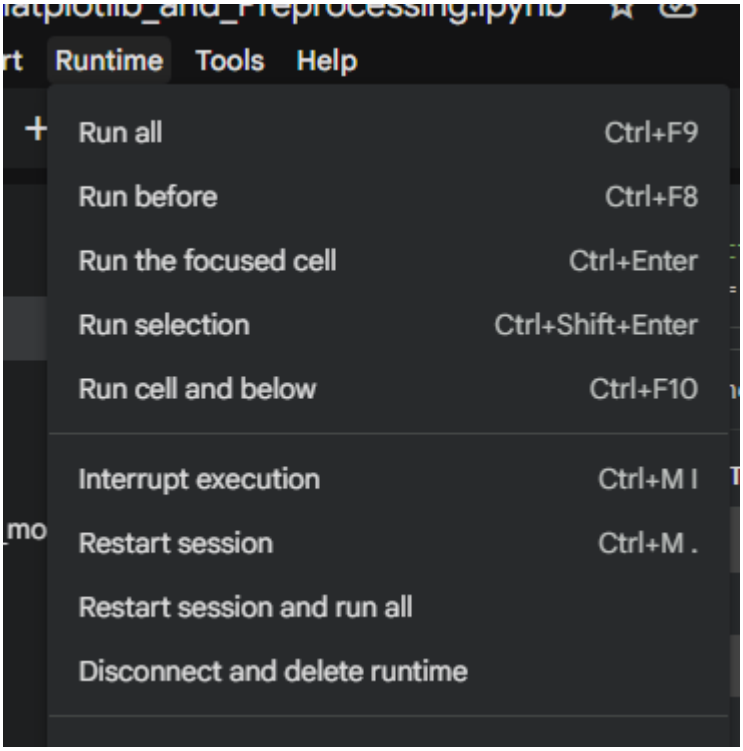



**Tips**

If data files they can be found in the file explorer on the left.




If you start encountering errors sometimes and not others, restart session is a useful command. A common reason for this is naming a variable the same as an existing function.


Runtime
Tools
Help
Run all Ctrl+F9
Run before Ctrl+F8
Run the focused cell Ctrl+Enter
Run selection Ctrl+Shift+Enter
Run cell and below Ctrl+F10
Interrupt execution Ctrl+M I
Restart session Ctrl+M .
Restart session and run all
Disconnect and delete runtime