Data Visualization

Agenda:

1. Python Matplotlib
2. Jupyter Notebook
3. Matplotlib Architecture
4. Questions

Here is a quick overview of what we'll cover today.

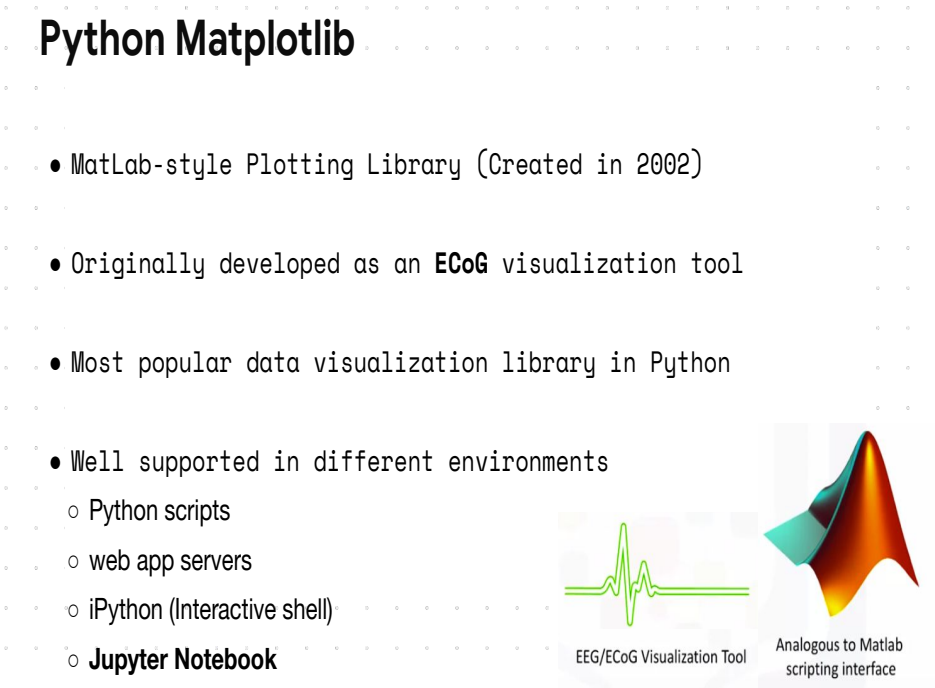
Python Matplotlib

John Hunter (Matplotlib Creator)

- Neurobiologist
- Part of a team analyzing **ElectroCorticoGraphy (ECoG) Signals**
  - **Electrocorticography** is the process of recording electrical activity in the brain
- The team
  - used a proprietary software (**MATLAB** based version) for analysis
  - had only one license and were taking turns in using it
- **John** replace the proprietary software with **Matplotlib**

A **neurobiologist**

- is a scientist who specializes in **neurobiology**,
  - which is the study of the nervous system from a biological perspective
- is typically a research scientist focused on unraveling the underlying biological mechanisms
- investigate the structure, function, development, and pathology of the nervous system



Python Matplotlib

- equipped with a scripting interface (**pyplot**) for quick and easy generation of graphics (like MATLAB)

- **ElectroCorticoGraphy (ECoG**)
- **ElectroEncephaloGraphy (EEG)**

Jupyter Notebook

Jupyter Notebook

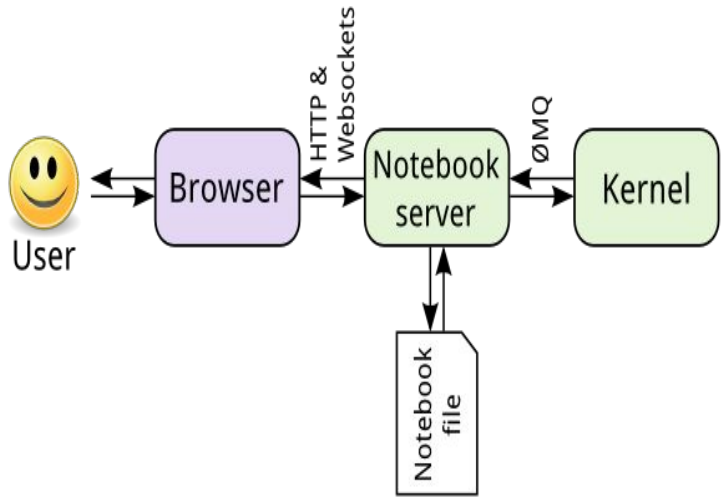
- open source web app
- allows to create & share documents that contain code and text
- spun off from **iPython** in **2014**

- **Jupyter name** is a reference to three programming languages:
  - **Ju**lia
  - **Py**thon
  - **R**
- **Jupyter logo**
  - homage to **Galileo**'s discovery of the **moons of Jupiter**
  - documented in **notebooks** attributed to **Galileo**

Jupyter Notebook

- grew out of the IPython project started by Fernando Perez
- **IPython** is an interactive shell, similar to the normal Python shell but with great features like syntax highlighting

Jupyter Notebook Workflow



https://ipython.org/ipython-doc/3/development/how_ipython_works.html

How Notebooks Work

1. User connect to the **server** through browser and the notebook is rendered as a web app (the notebook in the browser)
2. Code written in the web app is sent through the **server** to the IPython kernel (an IPython app running in the background)
3. The kernel runs the code and sends it back to the **server**, then any output is rendered back in the browser
4. When you save the notebook, it is written to the **server** as a JSON file with a .ipynb file extension

Jupyter Notebook support R & Julia

- The great part of this architecture is that the kernel doesn't need to run Python
- Since the notebook and the kernel are separate, code in any language can be sent between them
- **Ex**: two of the earlier non-Python kernels were for **R** and **Julia**
  - With an **R kernel**, code written in R will be sent to the R kernel where it is executed

- https://github.com/jupyter/jupyter/wiki/Jupyter-kernels
- https://www.r-project.org
- http://julialang.org

Jupyter Notebook can be accessed remotely

- Another benefit is that the server can be run anywhere and accessed via the internet

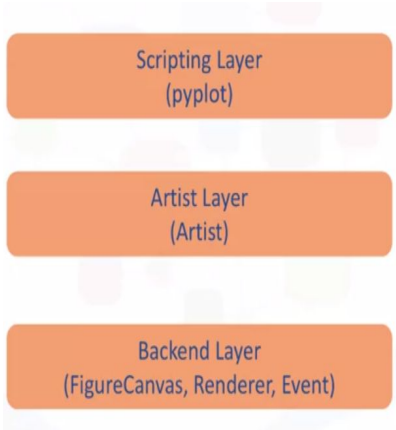
- You could also set up a server on a remote machine or cloud instance like Amazon's EC2.
  - Then, you can access the notebooks in your browser from anywhere in the world.

https://jupyter-notebook.readthedocs.io/en/stable/

Matplotlib Architecture

Matplotlib Architecture

1. **Back-end Layer**
2. **Artist Layer**
   - appropriate programming paradigm for
     - web app server
     - UI app
     - script to be shared with others
3. **Scripting Layer** (idea from MATLAB)
   - appropriate layer for everyday purposes
   - lighter interface to simplify common tasks
   - for a quick and easy generation of plots



- **Matplotlib**
  - is a Python library for data visualization
  - deeply integrated and commonly used with NumPy (a numerical mathematical library)

Matplotlib Architecture: 1) Back-end Layer

has built-in classes, such as:

1. **FigureCanvas:** *matplotlib.backend_bases.FigureCanvasBase*
   - defines and encompasses the area into which the figure is drawn
2. **Renderer:** *matplotlib.backend_bases.RendererBase*
   - knows how to draw (generate image) on the **FigureCanvas**
3. **Event:** *matplotlib.backend_bases.Event*
   - handles user inputs such as keyboard strokes and mouse clicks

Canvas

- defines and encompasses the area into which the figure is drawn

Renderer https://www.techopedia.com/definition/9163/rendering

- Rendering is the process involved in the generation of a 2D or 3D image
- Rendering is mostly used in
  - architectural designs
  - video games, and animated movies
  - simulators
  - TV special effects
  - design visualization

Matplotlib Architecture: 2) Artist Layer

- Contains one main abstract class (the **Artist**)

- **Artist**
  - knows how to use the **Renderer** to draw (put ink) on the **FigureCanvas**
  - https://github.dev/matplotlib/matplotlib/blob/main/lib/matplotlib/artist.pyi

- Everything we see on a **Matplotlib figure** is an **Artist instance**
  - **Example**: title, lines, tick labels, images, …
  - all of them correspond to an individual **Artist instance**

https://www.geeksforgeeks.org/matplotlib-artist-artist-draw-in-python

- **Artist class**
  - There is an Abstract base class
  - All visible elements in a figure (objects that render into a FigureCanvas) are subclasses of Artist
- **Artist** knows how to take the Renderer and use it to put ink on FigureCanvas

Matplotlib Architecture: 2) Artist Layer Types

1. **Primitive Artist**: as Line, Rectangle, Circle, Text

2. **Composite Artist:** may contain other **Artists**
   - **Example 1: Figure Artist** https://github.dev/matplotlib/matplotlib/blob/main/lib/matplotlib/figure.py
     - top-level Matplotlib object
     - contains and manages all of the elements in a given graphic
   - **Example 2: Axes Artist** https://github.dev/matplotlib/matplotlib/blob/main/lib/matplotlib/axes/_axes.py
     - most important Composite Artist
     - where most of the plotting methods are defined
     - including methods to create/manipulate ticks, axis lines, grid, background
   - **Other Examples: Tick Artist**

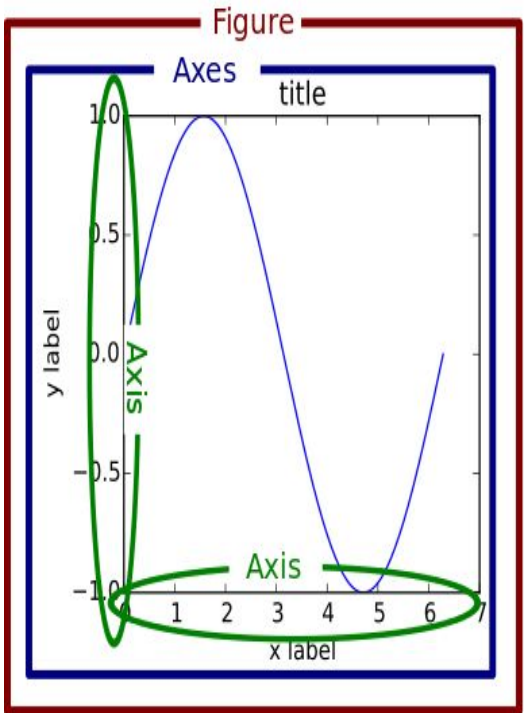
Also, there is also the Axis class

- https://github.dev/matplotlib/matplotlib/blob/main/lib/matplotlib/axis.py

Axes

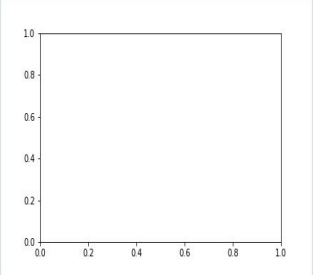
- The plotting area,
  - including any axis
- Don't mean plural of **Axis**

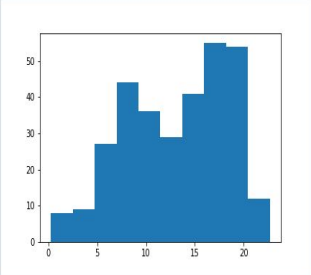
- When pronounced with short e
  - axes is the plural of axe
- When pronounced with long e
  - axes is the plural of axis



https://grammarist.com/homophones/axis-vs-axes

Figure and Axes





Create a Figure, add an Axes to the Figure... then plot the data inside that Axes!

Matplotlib Architecture: 3) Scripting Layer

- Developed for scientists who are not professional programmers

- Essentially the **Matplotlib.pyplot** that automates:
  - defining **FigureCanvas**
  - defining **Artist**
  - connecting **Artist** with **FigureCanvas**
  - https://github.dev/matplotlib/matplotlib/blob/main/lib/matplotlib/pyplot.py

- Compared to Layer 2 (**Artist Layer**), which is:
  - heavy and for developers
  - not for individuals who want to perform **quick EDA** of some data

https://www.activestate.com/resources/quick-reads/what-is-pyplot-in-matplotlib

- **Scripting Layer (pyplot)** is an API (Application Programming Interface) for Python matplotlib that effectively makes matplotlib a viable open source alternative to MATLAB.

Questions

Links

https://github.com/fcai-b/dv

References

1. https://www.coursera.org/learn/foundations-data
2. https://www.coursera.org/learn/what-is-datascience
3. https://www.coursera.org/learn/python-for-data-visualization