OHBM 2025 Educational; From Code to Visualization: Reproducible Pipelines for Neuroimaging Research

Sina Mansour L.

Jun 09, 2025

CONTENTS

From Code to Visualization

Author/Presenter:
Sina Mansour L., Ph.D.
National University of Singapore & The University of Melbourne

Note

This Jupyter Book contains supporting material for the session titled *"From Code to Visualization: Reproducible Pipelines for Neuroimaging Research"* that was part of an educational course on "Maximizing scientific efficiency through sustainability, reproducibility, and FAIRness", presented at the 2025 OHBM Annual Meeting in Brisbane.

Synopsis

How can neuroimaging researchers ensure their findings are not only robust but also **transparent** and **reproducible**?

This book guides you through building research workflows with reproducible **code** and **visualizations**, designed to accompany your manuscript and support open science.

Why Reproducibility?

Reproducibility helps:

- ✅ Maximize trust in your work
- Enable collaboration across teams
- Improve transparency and accountability

We provide examples and practical tips to boost reproducibility:

In your Code:

- Write clear, shareable scripts
- Follow best practices for sharing code
- Use open notebooks for full pipeline visibility

In your Visualizations:

- Understand why reproducible figures matter
- Learn how to make visualizations easy to recreate
- Build scriptable visualizations for:

  - Cortical surface projections
  - Volumetric plots
  - Tractography results
  - Brain network diagrams
  - … & more!

Table of contents

Overview of the content of this book:

CHAPTER

ONE

❓ REPRODUCIBILITY IN NEUROIMAGING

This chapter aims to highlight the importance of reproducible research practices and briefly explain what will be covered in the rest of the book.

1.1 ❓ Why Care about Reproducibility?

- **Science builds on science**, without reproducibility, findings can't be trusted or extended.
- In computational neuroscience and neuroimaging, findings are often results of complex pipelines, built with multiple software tools, and parameter choices.
- Without access to code, even small analytical tweaks can become untraceable.

  "Science is a social enterprise: independent and collaborative groups work to accumulate knowledge as a public good."

  Munafo et al. (2017)[1]

1.2 ❓ What's at Stake?

- **Credibility** of research outputs.

  "The authors of research papers have no obligation to share their data and code, and I have no obligation to believe anything they write."

  Andrew Gelman (professor of statistics and political science at Columbia University)
- **Wasted time** taken to re-implement procedures reported in previous works.

1.3 ❓ Categorizing Reproducibility:

Botvinik-Nezer and Wager[2] identify three types of reproducibility:

1. **Analytical reproducibility**: Reproducing findings using the *same data* and *same methods*.
2. **Replicability**: Finding similar results in *independent datasets* using similar methods.
3. **Robustness to analytical variability**: Obtaining consistent results using *different analytical approaches*.

[1] Marcus R Munafò, Brian A Nosek, Dorothy VM Bishop, Katherine S Button, Christopher D Chambers, Nathalie Percie du Sert, Uri Simonsohn, Eric-Jan Wagenmakers, Jennifer J Ware, and John PA Ioannidis. A manifesto for reproducible science. *Nature human behaviour*, 1(1):0021, 2017.

[2] Rotem Botvinik-Nezer and Tor D Wager. Reproducibility in neuroimaging analysis: challenges and solutions. *Biological Psychiatry: Cognitive Neuroscience and Neuroimaging*, 8(8):780–788, 2023.

Tip

The goal of this session is to introduce practices for ensuring analytical reproducibility. This can serve as a foundation for achieving replicability and methodological robustness.

Gorgolewski and Poldrack[3] cover 3 major topics in open science (see Fig. 1.1), with implications for reproducibility:

1. **Data**: Access to the original data is required to examine analytical reproducibility.
2. **Code**: Access to the implementation scripts is also needed.
3. **Papers**: Access to documentations of methods, and interpretations of results.

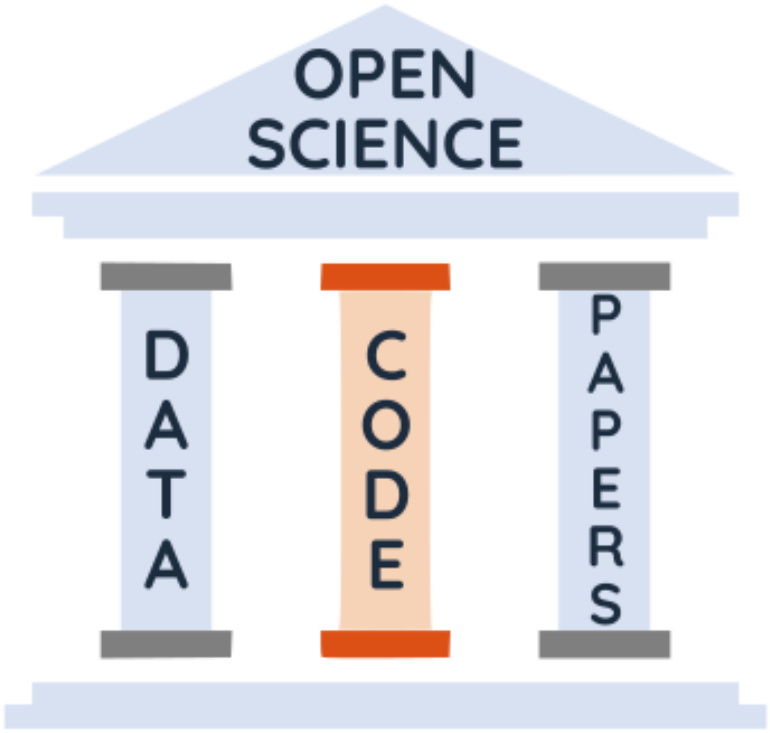


Fig. 1.1: Three pillars of open science.[3]

Tip

In this session, we'll cover topics addressing **code** openness.

[3] Krzysztof J Gorgolewski and Russell A Poldrack. A practical guide for improving transparency and reproducibility in neuroimaging research. *PLoS biology*, 14(7):e1002506, 2016.

1.4 ? Let's Start

Now that we've covered why reproducibility matters and what this session will include, let's jump in. We'll focus on two main topics:

- ***? Building Reproducible Analysis Pipelines***
- ***? Reproducible Scientific Visualizations***

1.5 ? References

CHAPTER

TWO

🔍 BUILDING REPRODUCIBLE ANALYSIS PIPELINES

This chapter aims to set the foundation for how to achieve analytical reproducibility, especially via **good coding practices**, **structured workflows**, and **tools** tailored for neuroimaging.

Key goals of this chapter:

- Show that reproducibility starts with how we write, document, and share code.
- Emphasize modular, readable, and shareable analysis pipelines.
- Provide examples on how to implement best practices in sharing code.
- Introduce relevant tools that can help you in sharing your analysis code.

2.1 🔍1🔍 Coding Best Practices

Let's address the elephant in the room: many researchers hesitate to share their code, often out of fear of being judged for how it's written.

While some level of imposter syndrome is inevitable, there are simple, effective habits you can adopt to boost both your confidence and your code quality. These best practices don't require much extra time, and they can save you significant effort down the line.

Code sharing is no longer a luxury, it's increasingly expected, made easier by accessible tools and community-driven standards. Open science is gaining momentum, and with it, a welcome shift: code doesn't need to be flawless to be valuable. In fact, any effort to share code is generally appreciated far more than not sharing anything at all.

In this chapter, we'll go over a set of practical recommendations, drawn from widely accepted coding standards—that can help you prepare code you'll feel comfortable sharing.

Not every point will apply to every project, but most will. Aim to adopt the practices that best fit your needs and workflow.

2.1.1 🔍 Small Steps Toward Better Coding Practices

In the ensuing sections, we'll dive through things you could do make your code better for sharing:

Commenting

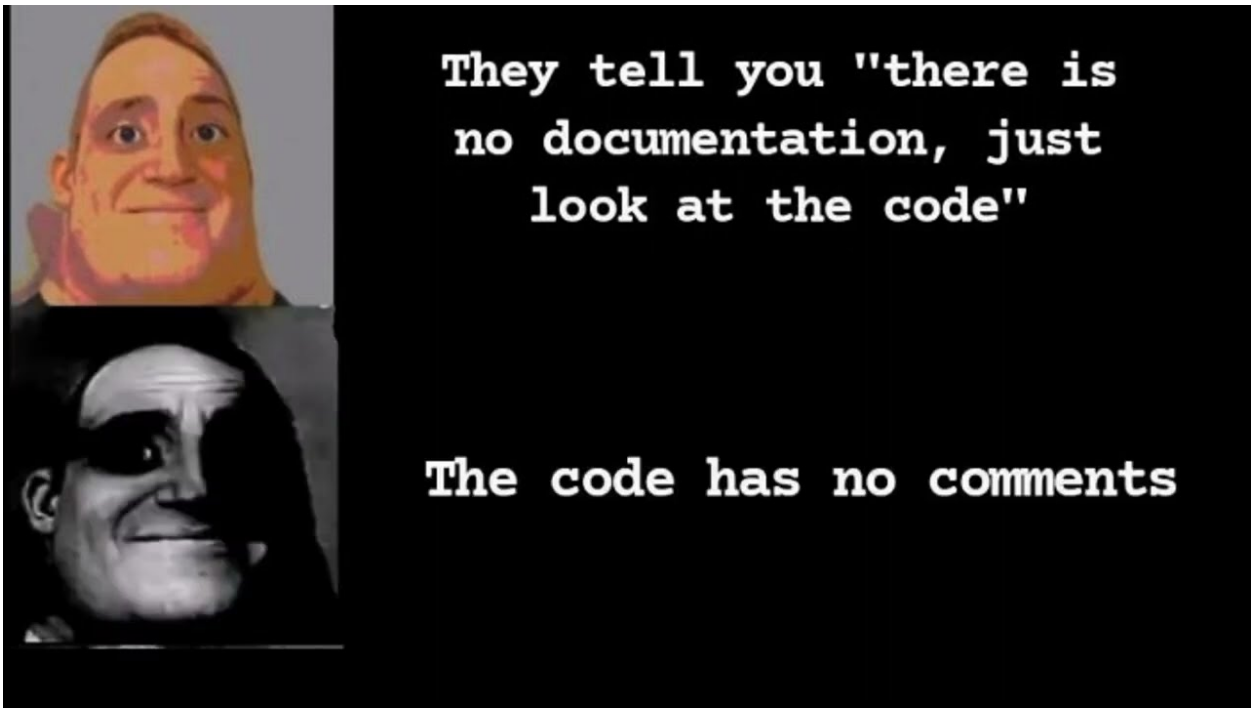


Fig. 2.1: Comments are essential for making code shareable and understandable.

- **Top-level script comments:** At the top of each executable script, include a short explanatory comment describing how it should be run. No matter how brief, this should include at least one example of expected usage (e.g., a sample command line call).
- **Inline comments:** Throughout your code, include comments explaining what key blocks or lines are doing. This isn't just for others—it'll help your future self, too!

Variables

- **Avoid hard-coding values.** Instead of embedding parameters directly into your code (e.g., file paths, thresholds, or flags), assign them to variables.
- **Centralize configurable parameters.** Define variables that others might need to modify, such as input/output paths or settings, at the top of your script. This makes your code easier to read, test, and reuse.

Example

Here's an example of what **not** to do:

```
if p_value < 0.05:
    ...
```

A better approach is to define parameters as variables at the top of your script:

```
# at the top of the script
p_value_threshold = 0.05

# further down the script
if p_value < p_value_threshold:
    ...
```

? Functions

- **Functions, functions, functions:** Functions are reusable building blocks. Break your code into logical, reusable functions wherever possible.
- A good rule of thumb: functions should be shorter than a page (~60 lines) and do one thing well.
- If a code block appears more than twice, it's probably worth turning into a function.

? Eliminate Duplication

- **Within your own code:** Use functions to avoid copy-pasting logic.
- **Beyond your code:** Don't reinvent the wheel! Most languages offer high-quality libraries for common tasks. Before building from scratch, check if a well-maintained package already solves your problem.

> **Warning**
>
> ? Be cautious when using new or experimental packages. Test imported functions before integrating them into your workflow.

? Naming Matters

- **Use meaningful names:** Choose descriptive names for variables and functions. The broader their scope or importance, the more informative the name should be.
  - For loop counters, short names like `i` or `j` are fine.
  - For important variables or data structures, avoid cryptic one-letter names.

> **Tip**
>
> **Tab Completion:** Most modern text editors support tab completion, so long, descriptive names don't slow you down!

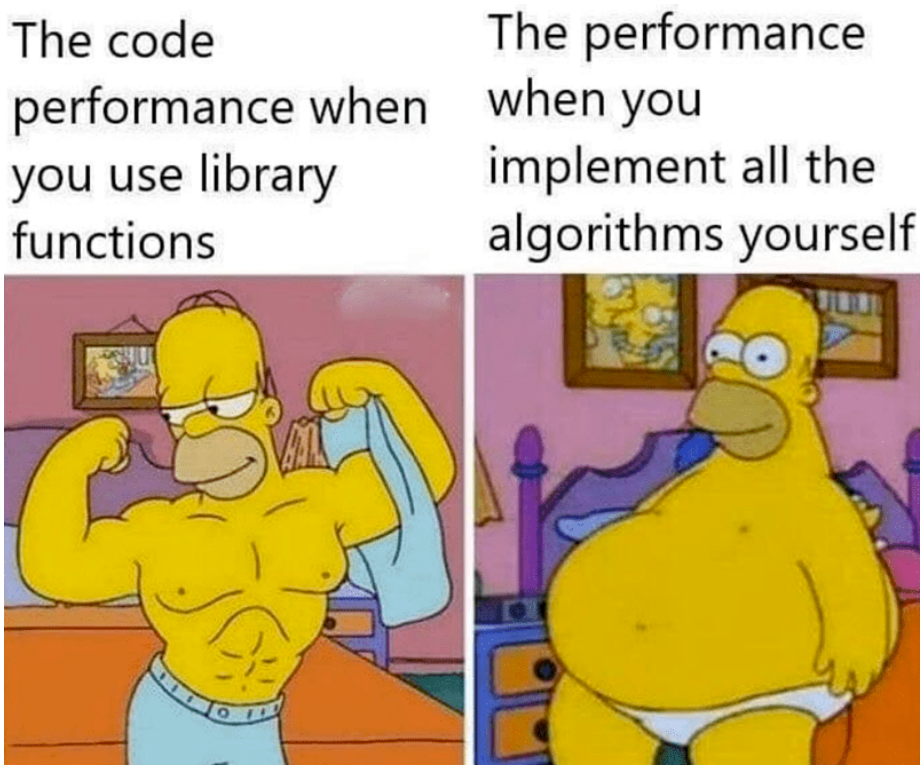


Fig. 2.2: The right use of library functions improves not just performance, but also the clarity of your script.

Fig. 2.3: When in doubt, opt for descriptive variable names over short but ambiguous ones.

Dependencies

- **List script requirements:** Include a `requirements.txt` file (or equivalent) to specify the dependencies needed to run your code.
- Alternatively, create a **Getting Started** section to your project's `README` file with clear setup instructions.

Avoid Commenting Out Code

- Don't toggle code behavior by commenting and uncommenting blocks. Instead, use `if`/`else` statements, configuration files, or flags for control flow.

Fig. 2.4: Avoid using comments to toggle code on or off—there are better ways.

Archive Your Code

- If you want to make your code citable and discoverable, consider archiving it on a DOI-issuing repository. Zenodo integrates seamlessly with GitHub, allowing you to create a citable snapshot of your project.

For further reading, you could check out the following resources:

1. Good enough practices in scientific computing[1]

[1] Greg Wilson, Jennifer Bryan, Karen Cranston, Justin Kitzes, Lex Nederbragt, and Tracy K Teal. Good enough practices in scientific computing. *PLoS computational biology*, 13(6):e1005510, 2017.

2. Open and reproducible neuroimaging: From study inception to publication[2]
3. The Turing Way's Guide for Reproducible Research
4. Best Practices for Writing Reproducible Code, an online course by Utrecht University.
5. British Ecological Society's Guide for Reproducible Code

2.1.2 ❓ References

2.2 ❓2❓ Using Notebooks Effectively

Not every research project results in a full software package. Sometimes, the core of your work is a collection of analyses or experiments that can be best expressed as a collection of code snippets, visualizations, and narrative. In these cases, preparing a full software repository might feel like overkill or a barrier to sharing. This chapter explains how Jupyter notebooks can provide a good solution to make your analysis code openly accessible.

2.2.1 ❓ Jupyter Notebooks to the Rescue

Jupyter notebooks offer a flexible, language-agnostic environment that blends code, rich text, and outputs in a single document. Whether you're using Python, R, Julia, or another language, notebooks let you organize your work interactively. Combined with version control systems like Git and platforms like GitHub, notebooks provide a simple but powerful way to share reproducible research.

When you follow best coding practices (covered in the previous section), your notebooks become readable, reusable, and easier to maintain. Plus, thanks to native support on platforms like GitHub, users can preview both your code and its outputs without needing to run anything locally.

---

[2] Guiomar Niso, Rotem Botvinik-Nezer, Stefan Appelhoff, Alejandro De La Vega, Oscar Esteban, Joset A Etzel, Karolina Finc, Melanie Ganz, Rémi Gau, Yaroslav O Halchenko, and others. Open and reproducible neuroimaging: from study inception to publication. *NeuroImage*, 263:119623, 2022.

Notebooks Support Multiple Languages

Jupyter is not limited to Python. It supports a wide variety of programming languages via "kernels," which are the computational engines that execute your code. Popular kernels include Python (ipython), R (IRkernel), Julia (IJulia), MATLAB (Matlab Integration for Jupyter) and many others. This flexibility allows you to write notebooks in the language best suited to your research domain or combine languages in the same project with multiple notebooks. (check here for an exhaustive list of available kernels)

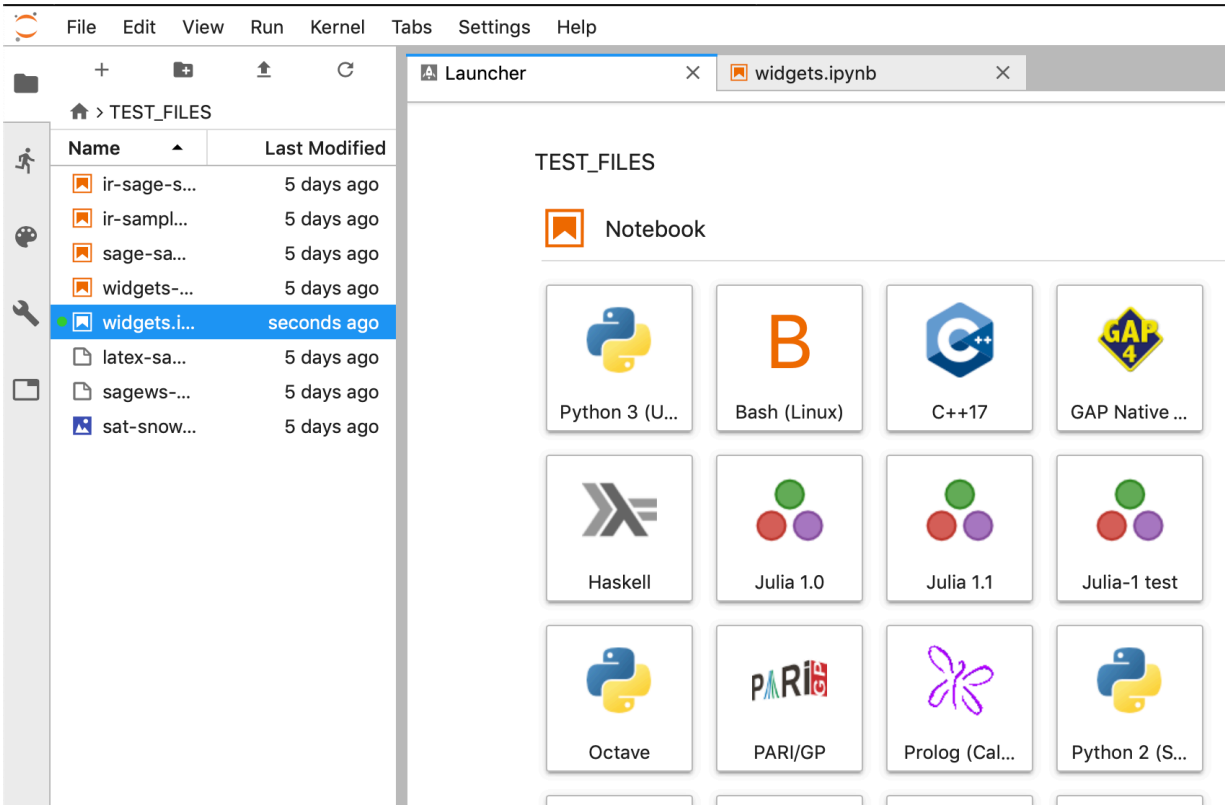


Fig. 2.5: Whatever programming language you use, there's a good chance it already has a Jupyter integration.

Native GitHub Rendering

GitHub automatically renders Jupyter notebooks, showing both the code cells and their outputs (plots, tables, text). This makes it easy for collaborators, reviewers, or readers to explore your work directly in their browsers without downloading or running any code. The integration boosts transparency and lowers the barrier for others to understand and build upon your results.

Integration with Live Execution Environments

One of the great strengths of notebooks is how easily they can be linked to live execution platforms:

- **Google Colab**: Provides free cloud-hosted Jupyter environments with pre-installed libraries, GPU support, and seamless integration with GitHub. By simply adding a *Open in Colab* badge, others can run and modify your notebook instantly, no setup required (see here for detail).
- **Binder**: Offers a way to create custom, reproducible computing environments launched directly from your GitHub repository. Binder builds a Docker image with all dependencies defined in your repo (e.g., `requirements.txt`, `environment.yml`), allowing users to interact with your notebooks live in their browsers. (check this step-by-step tutorial from The Turing Way to find out how)

These services make it simple to share not just static scripts, but fully executable research notebooks.

Recommended directory structure

To keep your project organized and simple, consider a clear directory layout:

```
project_directory/              # To be tracked by git
    ├── notebooks/              # All Jupyter notebooks (.ipynb files)
    │   ├── step_1/             # Divide you analysis into different steps
    │   │   └── analysis_1.ipynb
    │   ├── step_2/
    │   │   └── analysis_2.ipynb
    ¦   ¦
    │   └── README.md           # Overview or instructions for notebooks
    ├── data/                   # Raw/processed data files
    ├── scripts/                # [OPTIONAL] Standalone scripts and utility␣
↪functions
    ├── docs/                   # [OPTIONAL] Documentation
    ├── environment.yml         # or requirements.txt (for dependencies)
    └── README.md               # Project overview and instructions
```

This structure separates notebooks from scripts and data, making your project easier to manage and maintain. By adopting this layout from the beginning, you can seamlessly track your progress with Git. When your project is ready for publication, you can simply make the repository public—sharing open code (and potentially data) alongside your manuscript.

Additional Points to Consider

- **Clear Narrative and Modularization**: Keep your notebooks readable by using markdown cells to explain your analysis steps, and consider modularizing complex code into external scripts or functions imported into the notebook.
- **Avoid Large Outputs in Notebooks**: To keep notebooks lightweight and fast to load, avoid embedding very large data outputs or images. Instead, save large results externally and reference them.
- **Use Git LFS for large files**: If your notebooks or data files are large, consider using **Git Large File Storage (LFS)** to manage them efficiently within your repository.

CHAPTER

THREE

REPRODUCIBLE SCIENTIFIC VISUALIZATIONS

This chapter introduces the topic of **programmatic scientific visualizations**, highlights their critical role in **open and reproducible scientific practices**, and provides practical examples and recipes that can be applied in day-to-day research workflows

Key goals of this chapter:

- Emphasize the importance of making visualizations reproducible and shareable.
- Cover various resources, tools, and libraries that support reproducible scientific visualizations.
- Provide concrete examples of reproducible code for generating common types of visualizations frequently encountered in computational neuroscience and neuroimaging research.

3.1 1 Programmatic Visualizations

3.1.1 Acknowledgment

This chapter draws inspiration from a talk by Dr. Sid Chopra, whose insights closely align with the themes I intended to cover here. Several key ideas are adapted, with his kind permission, from that presentation. I'm grateful to Dr. Chopra for generously allowing me to share and expand on them.

3.1.2 Why Visualizations Matter

> If I can't picture it, I can't understand it.
>
> *Albert Einstein*

In scientific research, figures often outlast the text. They are what readers remember, what journalists spotlight, and sometimes even what enters public imagination and pop culture.

The Impact of Visualization

A powerful example is the widely circulated brain connectivity figure from a psychedelic study by Petri *et al.*[1] (see Fig. 3.1). Beyond communicating the core findings, this figure captured the attention of numerous media outlets and became emblematic of the study itself.

Its influence extended well beyond academia, appearing in TEDx talks (in Oxford, Warwick, Aarhus, and Varna) and media stories including Wired, Vox, Mind Medicine Australia, Medium, 7 news, and Business Insider.

[1] Giovanni Petri, Paul Expert, Federico Turkheimer, Robin Carhart-Harris, David Nutt, Peter J Hellyer, and Francesco Vaccarino. Homological scaffolds of brain functional networks. *Journal of The Royal Society Interface*, 11(101):20140873, 2014.

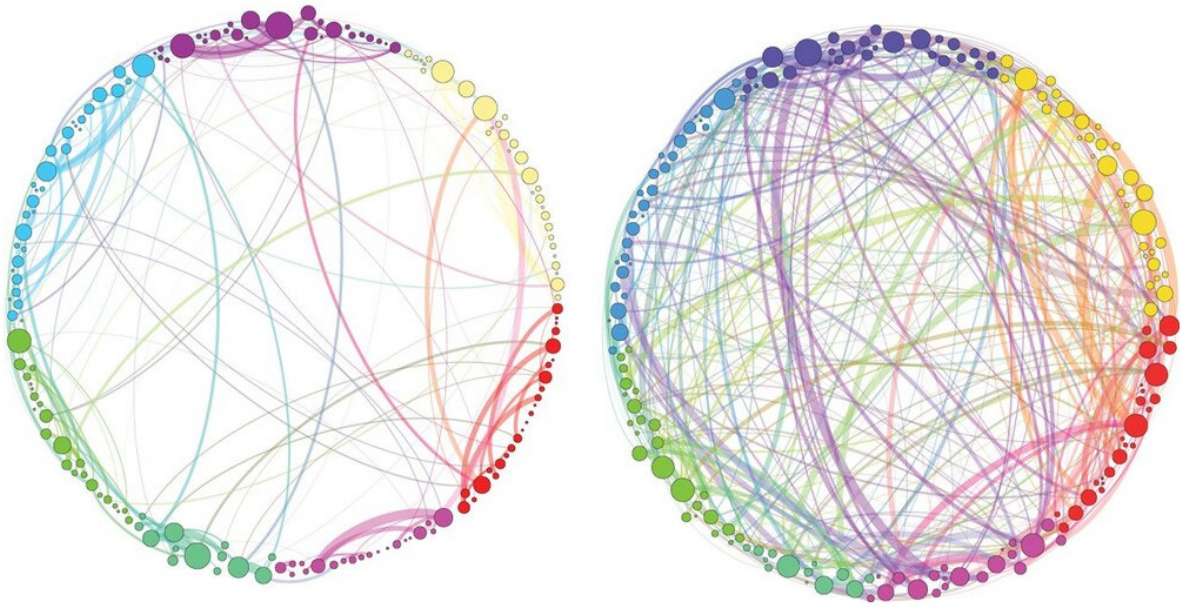


Fig. 3.1: Impact of Psilocybin on brain connectivity as illustrated by by Petri *et al.*[Page 15, 1].

This figure became **the public face of the study**, an iconic example of how visualizations can amplify a paper's reach and cultural footprint. Yet despite its popularity, the original image remains difficult to reproduce, as standards for sharing visualizations openly were not commonplace in 2014.

Fortunately, the field has evolved. Visualization science now emphasizes openness and reproducibility, supported by a growing ecosystem of tools designed to create sharable, transparent, and programmatically generated figures.

3.1.3 ⍰ Code- vs. GUI-Based Visualizations

There are two common paradigms for creating scientific visualizations:

- **GUI-based visualizations**: Created through graphical user interfaces, often involving manual adjustments, screenshots, or exports. No coding is required.
- **Code-based visualizations**: Built using scripts where the figure's core elements are generated programmatically, with minimal manual adjustments (e.g. for annotations).

GUI-based workflows can be fast and intuitive, giving researchers the freedom to tweak visuals until they "look right." However, this manual process makes it difficult to reproduce the figure, especially if the original settings are undocumented or the utilized tool is not mentioned.

In contrast, **code-based visualizations** are inherently more reproducible. Since the figure is generated from a script, it can be re-run by anyone, on a different machine, with updated parameters and data. This makes code-based approaches a foundational tool for open, transparent, and reusable science.

3.1.4 ? Why Use Programmatic Visualizations?

Beyond reproducibility, programmatic visualizations offer several powerful advantages:

- **Flexibility**: Seamlessly tweak styles, input data, visualization parameters, or output formats.
- **Scalability**: Efficiently generate dozens or hundreds of plots in a single script; ideal for individual-level reports, making animations, iterative experiments, or sensitivity analyses.
- **Interactivity**: Tools like Plotly, Bokeh, and Altair support interactive plots that can be embedded in notebooks or the web.
- **Version control**: Code-generated figures can be tracked with Git alongside your analysis pipeline.
- **Automation**: Integrate visualizations directly into your analysis pipelines, no manual export needed.
- **Data integrity**: The connection between data and final figure is transparent and auditable, leaving no hidden/unknown steps.
- **Skill reusability**: While the learning curve can be steep initially, the skills and code you develop are reusable across projects, yielding compounding returns.

These strengths make code-based visualization not just a stylistic preference, but a cornerstone of reproducible and transparent research, and an essential skill in any researcher's toolkit.

3.1.5 ? What Next?

In the next section, we'll introduce a range of tools, libraries, and templates that make code-based visualizations easier and more powerful. We'll explore how these tools integrate with Jupyter notebooks and neuroimaging file formats, enabling you to share interactive, reproducible, and publication-ready figures as part of your research outputs.

3.1.6 ? References

3.2 ?2? Visualization Tools Overview

This chapter offers a broad overview of tools and libraries that support the programmatic creation of publication-quality figures. While I don't consider myself an expert in scientific visualization, I've explored a variety of tools in my own research, and I hope this section provides a helpful starting point for others. Since my experience is primarily with Python, the tools discussed here are mostly Python-based. However, I've included links at the end of the chapter that point to similar resources for other programming environments.

Most tools are introduced only briefly, with links to more comprehensive guides. The goal is to highlight what's possible and to help you discover tools that might suit your own research needs.

The next two sections will cover general-purpose and neuroimaging-specific visualization libraries, respectively.

3.2.1 ⍰ General Visualization Tools

This section briefly introduces a set of Python libraries commonly used for general-purpose data visualization. These tools are widely applicable across scientific disciplines and can be used to generate high-quality plots for publications, presentations, and exploratory analysis.

Basic Plotting Libraries

Two of the most widely used libraries for data visualization in Python are **Matplotlib** and **Seaborn**:

- **Matplotlib** is a powerful and flexible low-level library for creating plots. It offers fine-grained control over every aspect of a figure.

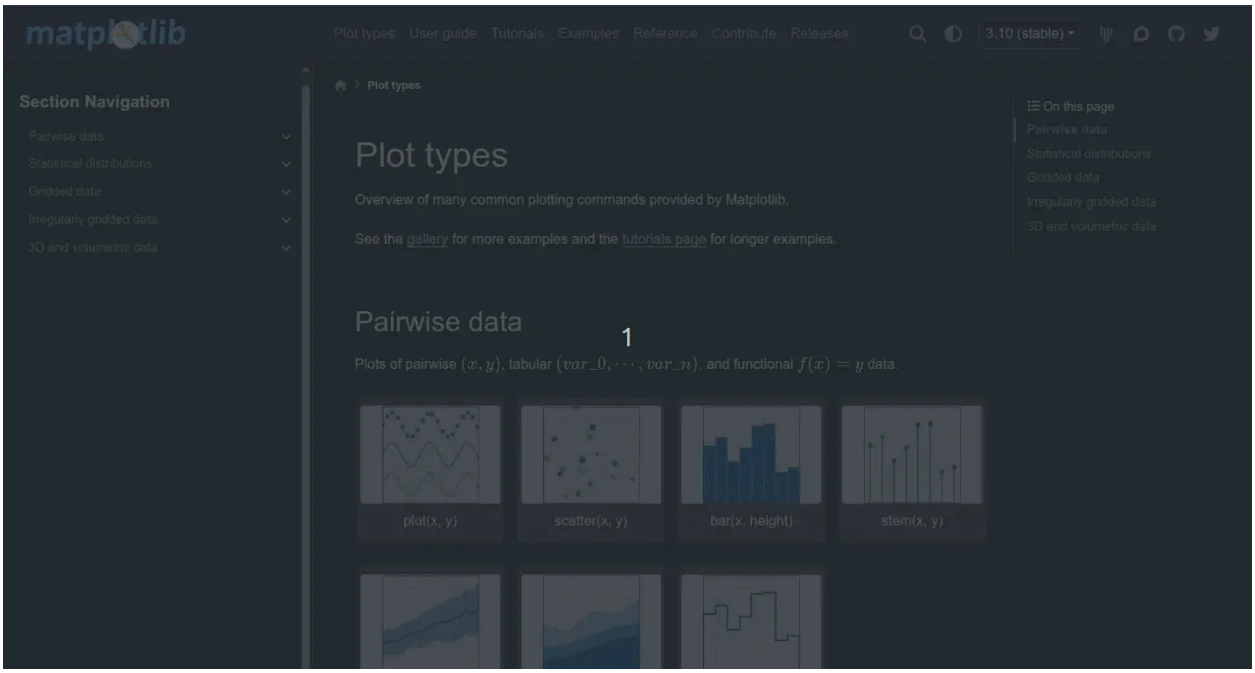


Fig. 3.2: A gallery of various plot types generated via **Matplotlib**.

- **Seaborn** is built on top of Matplotlib and provides a higher-level, more convenient API, particularly well-suited for statistical data visualization.

Fig. 3.3: A gallery of various plot types generated via **Seaborn**.

These libraries support a wide range of common plot types—including line plots, scatter plots, bar plots, histograms, box plots, heatmaps, violin plots, and more—which you can explore in detail through the linked gallery pages.

Some key features worth highlighting:

- ✅ **Multi-panel layouts**: You can compose complex, multi-panel figures entirely within Matplotlib (and Seaborn), making them suitable for direct inclusion in publications.

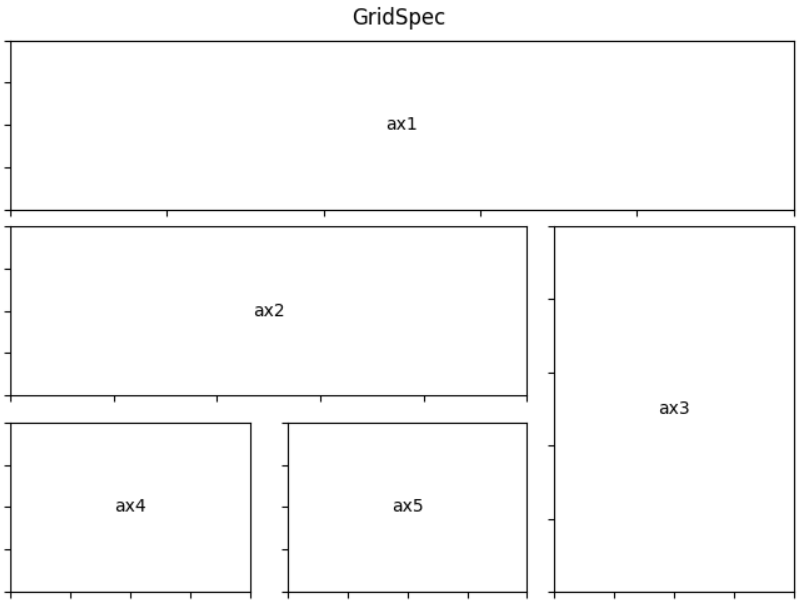


Fig. 3.4: Matplotlib's `Gridspec` provides a flexible way to make multiple panels in a single figure.

- ✅ **Flexible output formats**: Export figures to various formats such as PNG, PDF, SVG, and EPS.
- ✅ **LaTeX rendering**: Seamless integration with LaTeX lets you render mathematical notation directly in your plots.
- ✅ **Integration with NumPy/Pandas**: Native support for working with common data structures.

We won't cover how to write basic plotting scripts here, but links to tutorials and documentation are provided at the end of this chapter.

Supporting Data Structures

In neuroimaging and other data-intensive fields, efficiently managing and transforming large datasets is essential. Several foundational Python libraries serve as the backbone of many visualization workflows:

- **NumPy**: Provides fast and efficient numerical operations on arrays and matrices.
- **Pandas**: Offers powerful data structures (like `DataFrame`) for tabular data, with built-in methods for handling missing data, filtering, grouping, and more.
- **Xarray**: Designed for N-dimensional labeled arrays (e.g., time × region × subject), making it especially useful for spatiotemporal datasets and more complex data formats.

Most visualization libraries can directly accept these structures as input, making them interoperable within scientific analysis pipelines. For instance, a `.csv` file can be loaded into a `pandas.DataFrame`, which can then be passed directly to Seaborn for visualization.

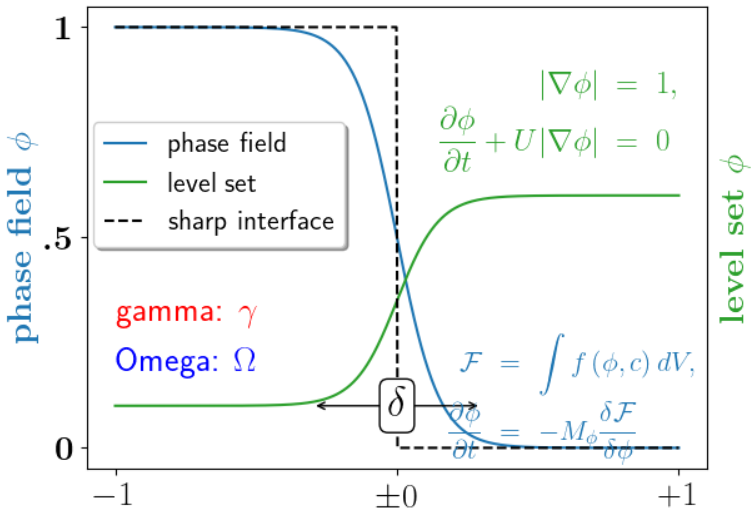


Fig. 3.5: Matplotlib has built-in support for latex, making it possible to integrate equations in figures.

Interactive Visualizations

While static plots are great for publications, interactive visualizations are invaluable for data exploration and sharing results in dynamic formats (e.g., web dashboards, notebooks). Several libraries support building rich, interactive graphics:

- **Plotly**: Offers intuitive syntax for interactive plots, with features like hover info, zoom, and click events.

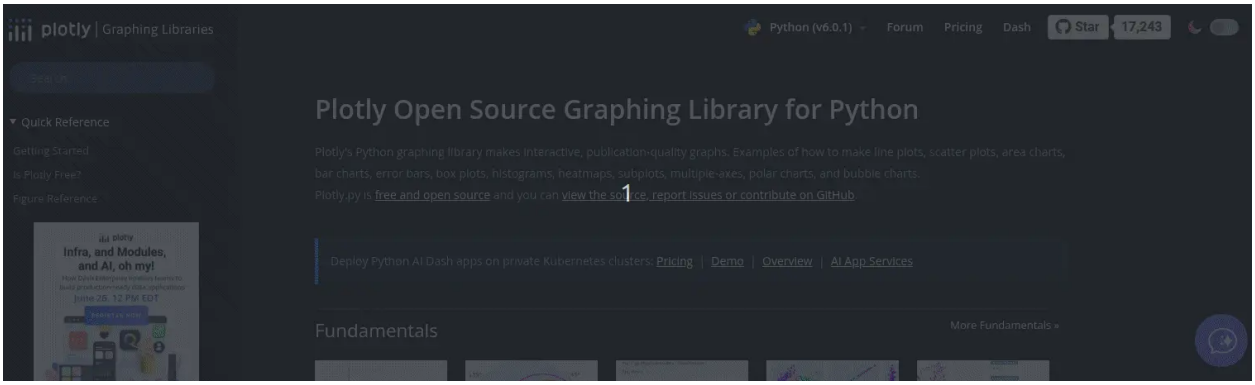


Fig. 3.6: A gallery of various plot types generated via **Plotly**.

- **Altair**: A declarative library built on the Vega-Lite grammar of graphics, great for producing concise and interactive charts.
- **Bokeh**: Ideal for building interactive dashboards or embedding visualizations in web applications.
- **Panel**: A versatile library for creating interactive apps and dashboards, supporting multiple plotting libraries (e.g., Bokeh, Plotly, Matplotlib, Altair) and well-suited for use in notebooks or standalone web apps.
- **ipywidgets**: Useful for adding interactivity within Jupyter notebooks, especially when combined with Matplotlib or Plotly.
- **Dash**: A framework (built on top of Plotly) for creating full web applications with interactive graphs and controls

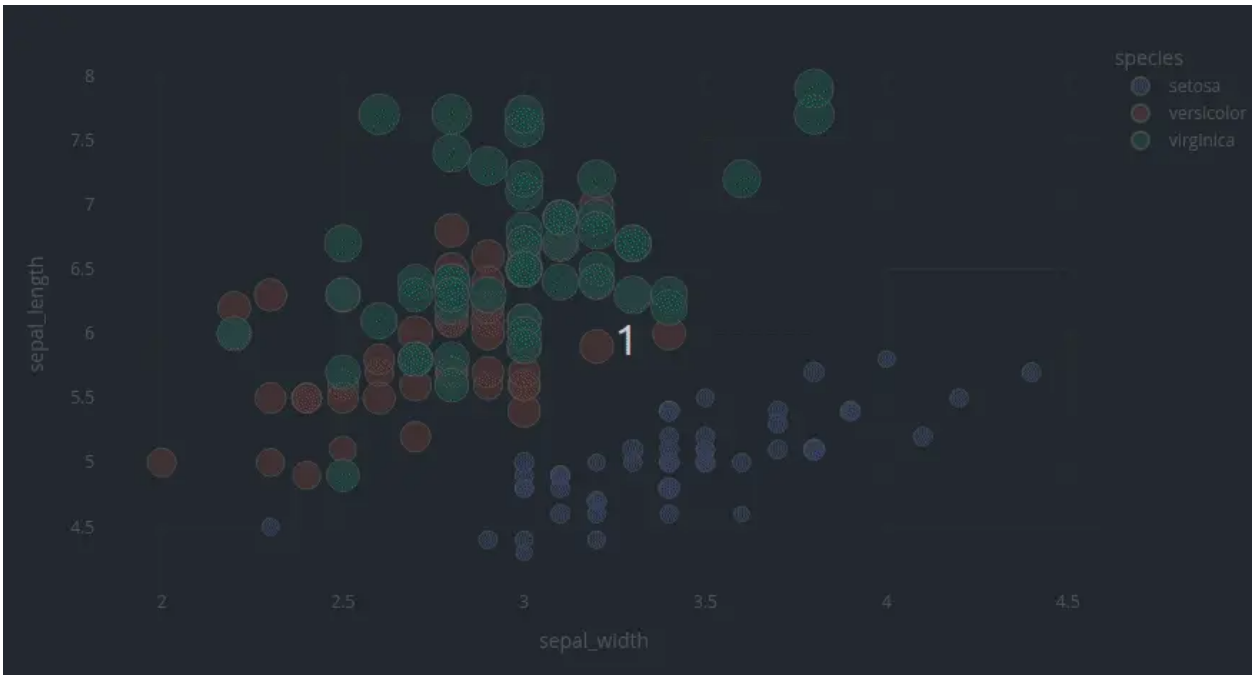


Fig. 3.7: **Plotly** can generate interactive visualizations.

using pure Python.

These tools are particularly useful in designing exploratory analysis, making interactive reports, and building reproducible visualizations inside notebooks.

Visualizing Large Datasets with Shaders

For very large datasets, such as scatter plots with many points, traditional plotting libraries can become unresponsive or fail to render efficiently. This is where GPU-accelerated, shader-based visualization becomes useful.

Datashader (by HoloViz) is specifically designed to render huge datasets by computing aggregate representations (with GPU support). Instead of plotting individual points, it computes and shades the density of data in image space, resulting in clean, informative visualizations even with billions of points.

Choosing Effective Colormaps

Colormaps are a fundamental component of scientific visualization, shaping how patterns and contrasts in data are perceived. Poor choices can obscure structure or introduce misleading gradients.

Key considerations:

- **Perceptual uniformity**: Changes in value should be perceived evenly across the range.
- **Data type**: Use sequential colormaps for ordered data, diverging for values around a central reference (e.g., ± change), and categorical for discrete labels.

Matplotlib and Seaborn offer a variety of useful palettes; for instance, `viridis`, `plasma`, and `cividis` are perceptually uniform. For a wider range of options, the **Colorcet** package provides a rich set of perceptually uniform and publication-ready colormaps designed with clarity and aesthetics in mind.

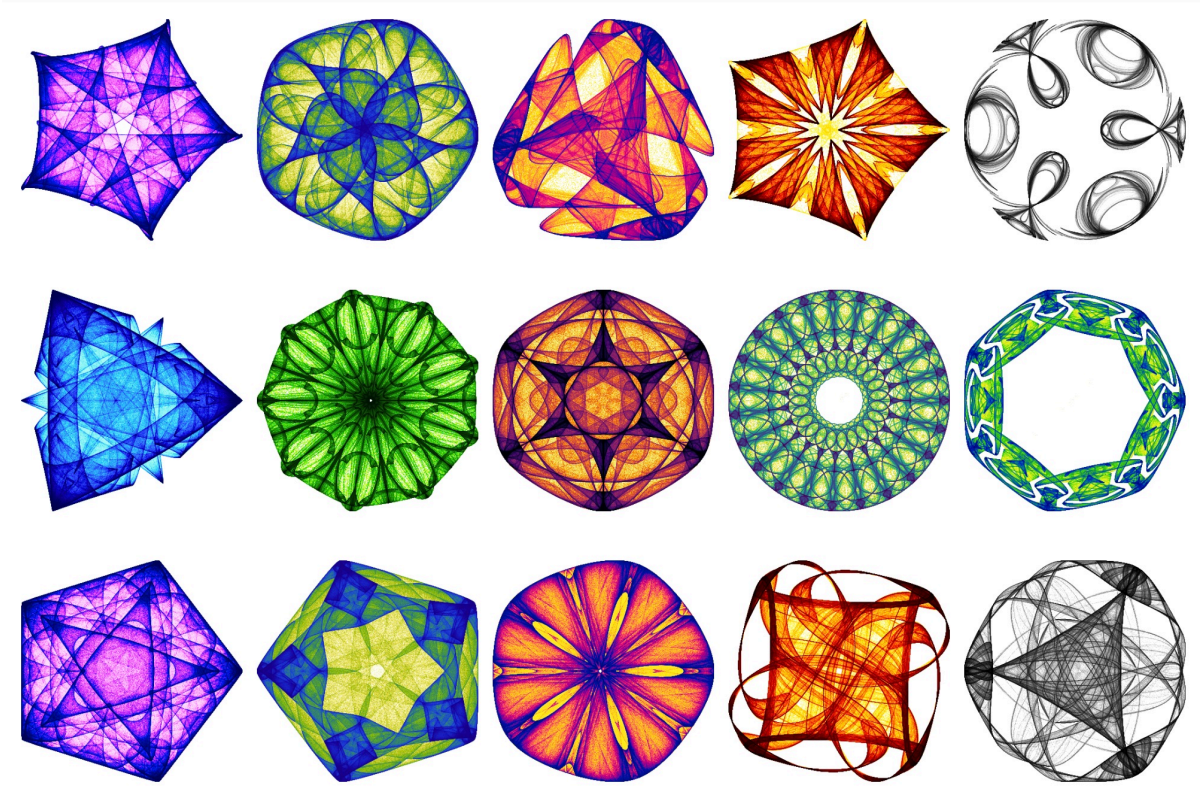


Fig. 3.8: Datashader can be used to visualize dynamical system attractors (with 10 million points each).

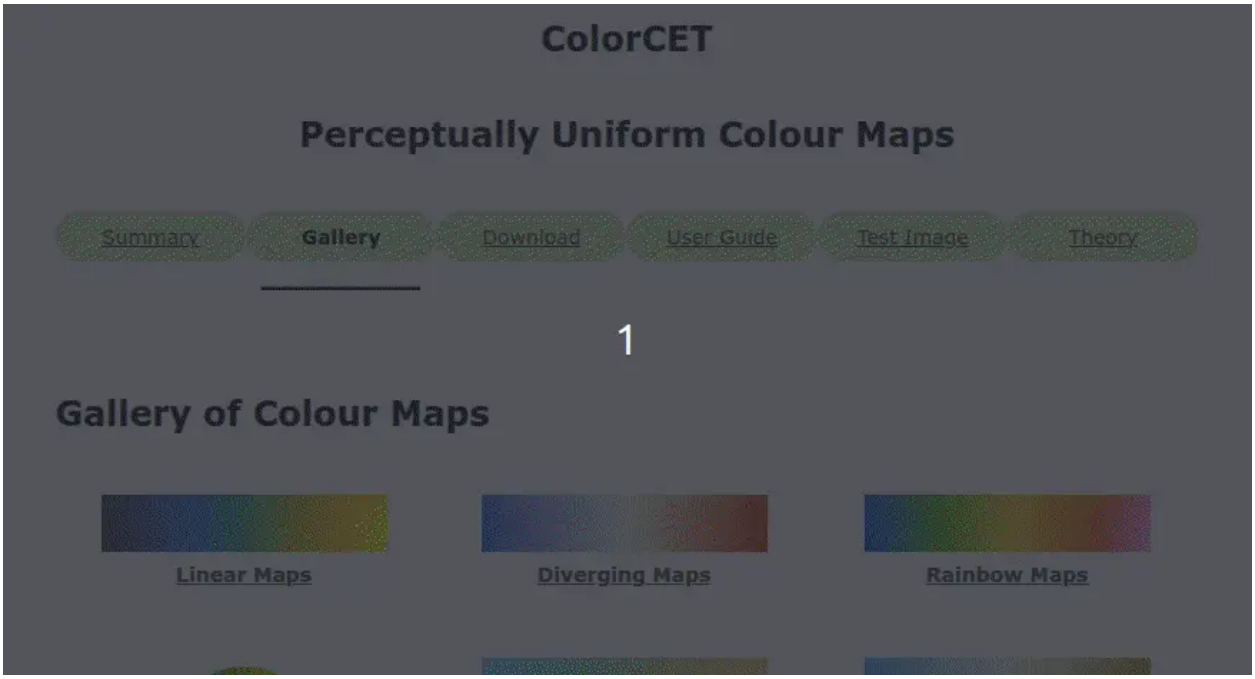


Fig. 3.9: The **Colorcet** package contains a wide array of perceptually uniform colormaps to suit different use cases.

Using appropriate colormaps not only improves figure readability and aesthetics, but also upholds scientific integrity by avoiding unintentional distortions in the data's visual representation.

3.2.2 ? Neuroimaging Visualization Tools

Having introduced a suite of general-purpose data visualization tools, we now turn to software packages that are specifically tailored for neuroscience and neuroimaging. These tools are designed to handle domain-specific data formats and support visualizations that are uniquely relevant to brain imaging research.

> **Note**
>
> This list includes several representative tools for each visualization type, but it is by no means exhaustive. We provide links to broader, community-maintained lists at the end of the section and encourage readers to explore online, as the neuroimaging ecosystem is constantly evolving.

Handling Neuroimaging Data

Before visualization, neuroimaging datasets must be loaded and decoded into memory. The following Python libraries provide robust support for a variety of brain imaging file formats:

- **NiBabel**: A foundational library for reading and writing many imaging file formats such as NIfTI, CIFTI, GIFTI, and FreeSurfer surfaces/labels.
- **DIPY**: A powerful library for diffusion MRI analysis and tractography, including support for DICOM/NIfTI I/O.

Volume Slice Rendering

A foundational form of neuroimaging visualization involves displaying anatomical or functional slices from 3D volumetric scans (e.g., T1-weighted MRI, statistical maps). These figures are ubiquitous in the literature for their clarity and ease of interpretation.

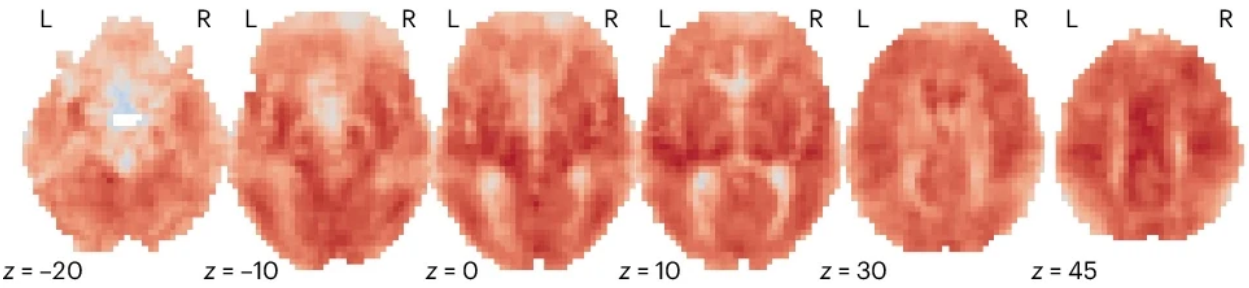


Fig. 3.10: An example of volume slice rendering, adapted from Bolt *et al.*[1].

? Python tools for generating volume slice renders:

- **NiBabel**
- **Nilearn**

[1] Taylor Bolt, Shiyu Wang, Jason S Nomi, Roni Setton, Benjamin P Gold, BT Yeo, J Jean Chen, Dante Picchioni, Jeff H Duyn, R Nathan Spreng, and others. Autonomic physiological coupling of the global fmri signal. *Nature Neuroscience*, pages 1–9, 2025.

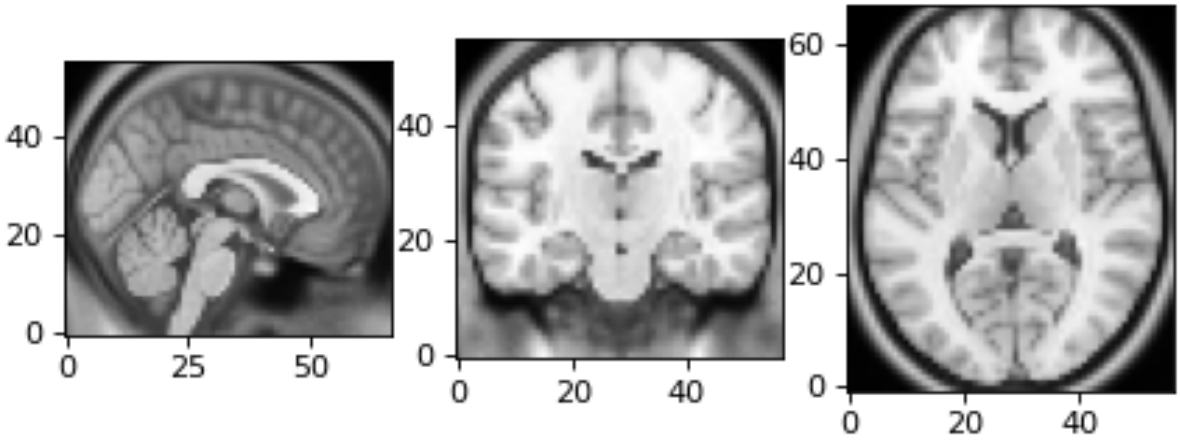


Fig. 3.11: **NiBabel** provides low-level access to volumetric data and can be used in combination with `matplotlib.pyplot.imshow` to generate slice visualizations..

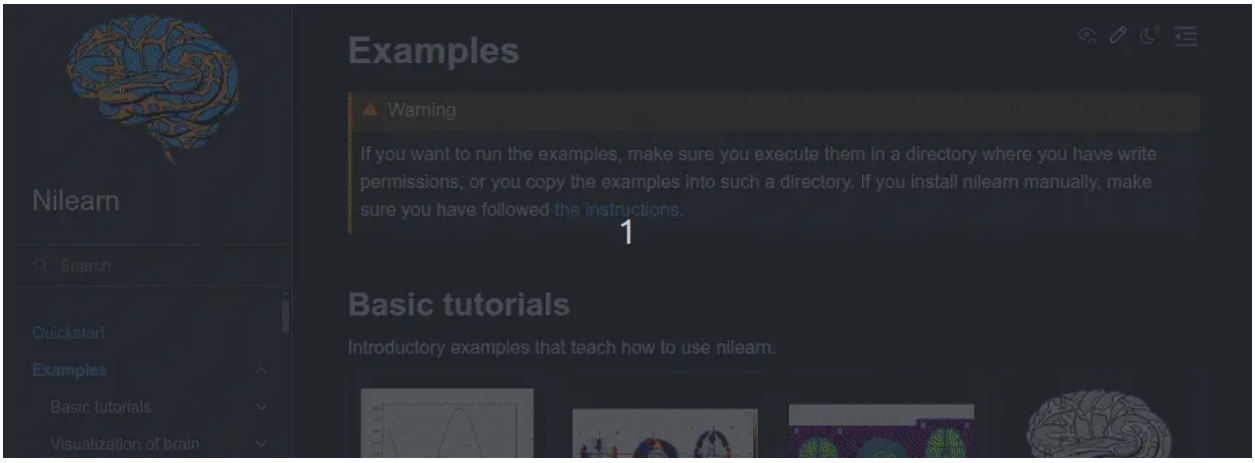


Fig. 3.12: **Nilearn** is a high-level library for statistical neuroimaging that includes built-in support for volume slice rendering and other visualization techniques.

- **nanslice** is a lightweight library for visualizing slices from 3D volumes.

Surface-based Visualizations

Surface-based visualizations render cortical metrics (e.g., thickness, functional activation) on a 3D model of the cortical sheet. These projections are particularly useful for visualizing data constrained to the cortical surface, offering an intuitive, continuous view of spatial patterns that might be obscured in standard volume slice renderings.

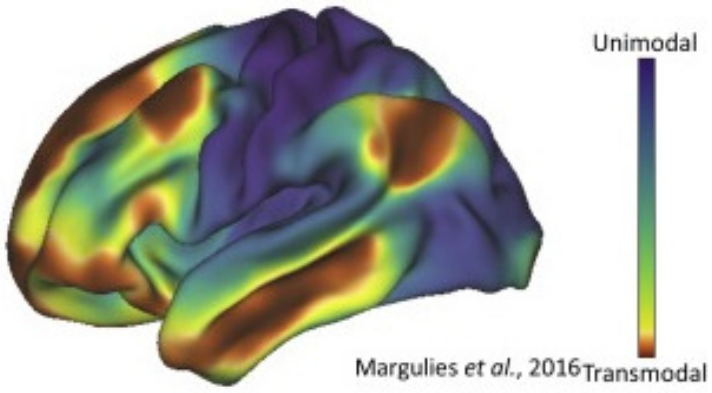


Fig. 3.13: An exemplary surface-based visualization depicting the principal functional gradient[2], adapted from Huntenburg *et al.*[3].

 Python tools for generating surface-based visualizations:

- **Cerebro Brain Viewer**

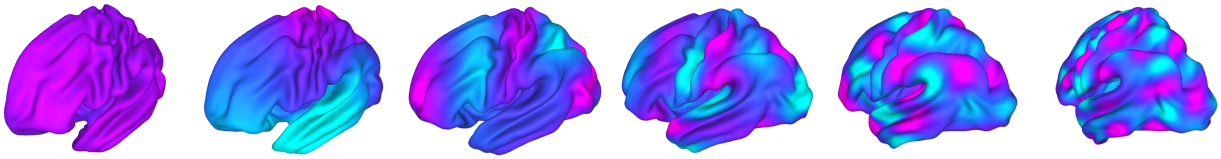


Fig. 3.14: A set of surface-based visualizations from **Cerebro Brain Viewer**.

- **Brainplotlib**
- **PySurfer**
- **SurfIce**

[2] Daniel S Margulies, Satrajit S Ghosh, Alexandros Goulas, Marcel Falkiewicz, Julia M Huntenburg, Georg Langs, Gleb Bezgin, Simon B Eickhoff, F Xavier Castellanos, Michael Petrides, and others. Situating the default-mode network along a principal gradient of macroscale cortical organization. *Proceedings of the National Academy of Sciences*, 113(44):12574–12579, 2016.

[3] Julia M Huntenburg, Pierre-Louis Bazin, and Daniel S Margulies. Large-scale gradients in human cortical organization. *Trends in cognitive sciences*, 22(1):21–31, 2018.

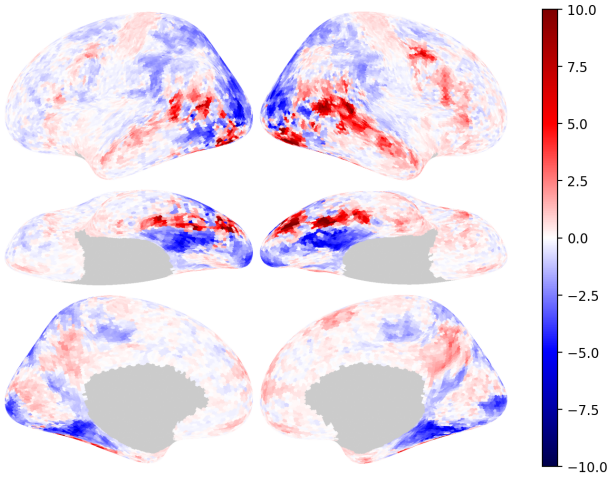


Fig. 3.15: An example surface-based visualizations from **Brainplotlib**.

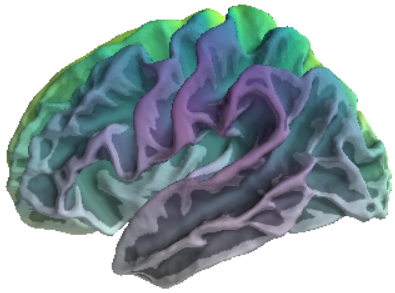


Fig. 3.16: An example surface-based visualizations from **PySurfer**.

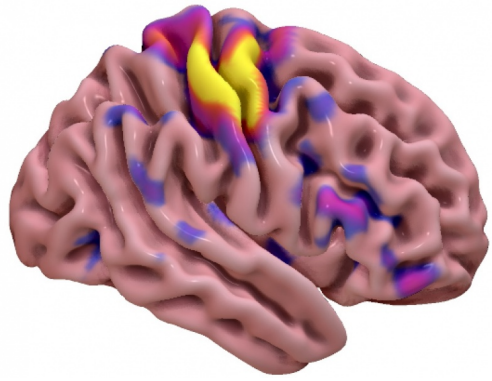


Fig. 3.17: An example surface-based visualizations from **SurfIce**.

Volume-to-Surface Transformation

When data originates in volumetric space (e.g., atlas-based ROIs, or a brain mask), it can be useful to project it onto the cortical surface for clearer spatial interpretation.

Fig. 3.18: An exemplary ROI to surface transformation of the Yale Brain Atlas, adapted from McGrath *et al.*[4].

 Python tools for generating volume to surface transformations:

- **Cerebro Brain Viewer**

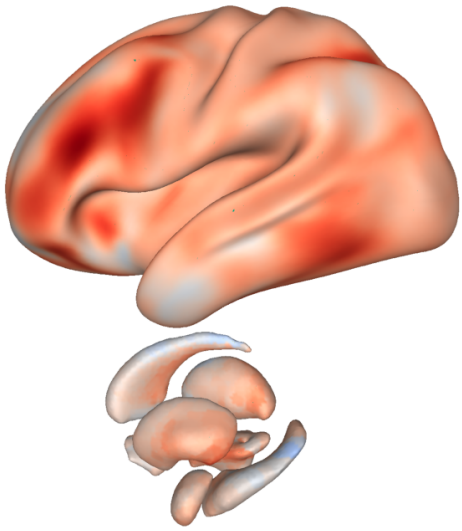


Fig. 3.19: An exemplary surface transformation of subcortical structures from **Cerebro Brain Viewer**.

- **SurfIce**

[4] Hari McGrath, Hitten P Zaveri, Evan Collins, Tamara Jafar, Omar Chishti, Sami Obaid, Alexander Ksendzovsky, Kun Wu, Xenophon Papademetris, and Dennis D Spencer. High-resolution cortical parcellation based on conserved brain landmarks for localization of multimodal data to the nearest centimeter. *Scientific reports*, 12(1):18778, 2022.

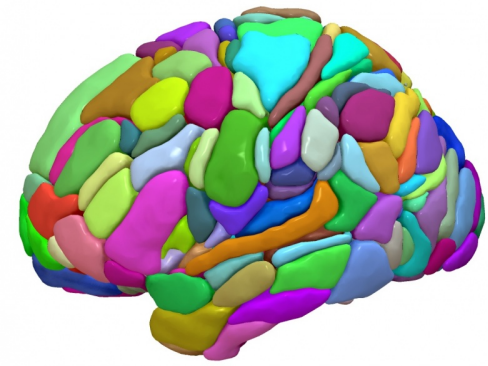


Fig. 3.20: An example surface-based visualizations of the AICHA template from **SurfIce**.

Tractography Visualization

Visualizing white matter fiber bundles from diffusion MRI is a key part of tractography-based studies. These plots often overlay streamlines on anatomical backdrops or 3D renderings of the brain.

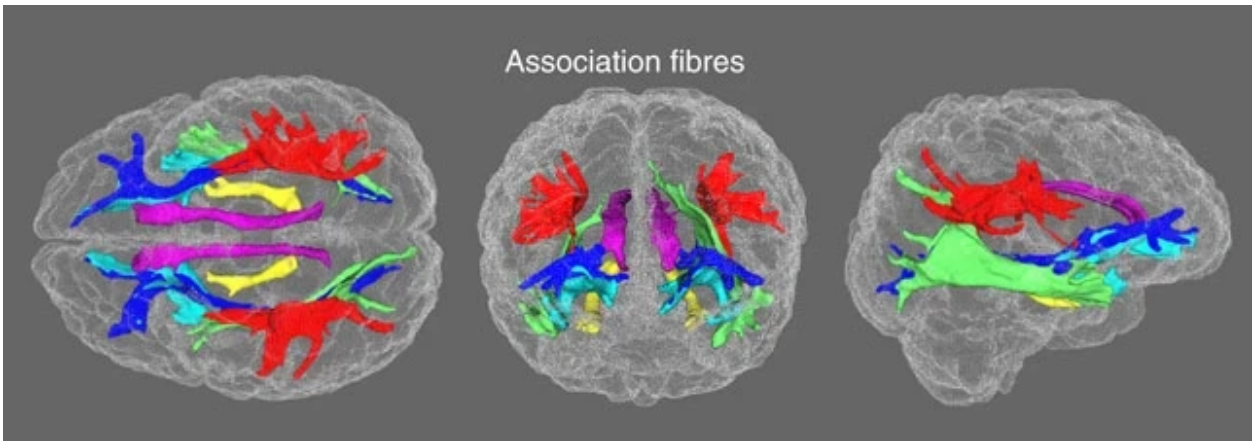


Fig. 3.21: An exemplary tractography visualization of the white-matter bundles, adapted from Cox *et al.*[5].

 Python tools for tractography visualization:

- **DIPY**
- **SurfIce**

[5] Simon R Cox, Stuart J Ritchie, Elliot M Tucker-Drob, David C Liewald, Saskia P Hagenaars, Gail Davies, Joanna M Wardlaw, Catharine R Gale, Mark E Bastin, and Ian J Deary. Ageing and brain white matter structure in 3,513 uk biobank participants. *Nature communications*, 7(1):13629, 2016.

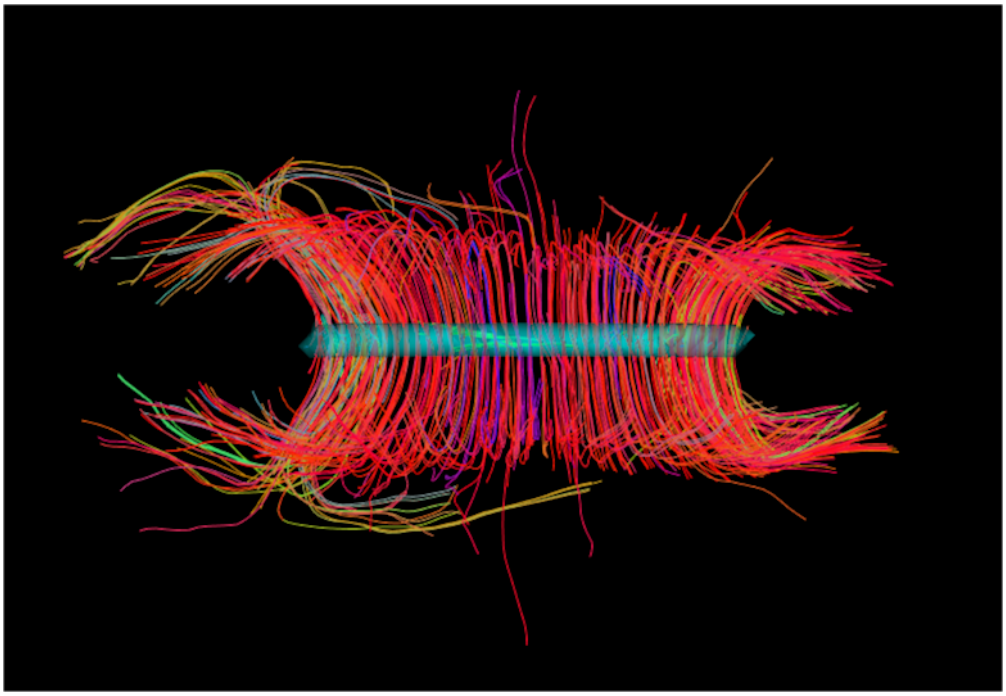


Fig. 3.22: An example visualization of tractography streamlines along the corpus callosum via **DIPY**.

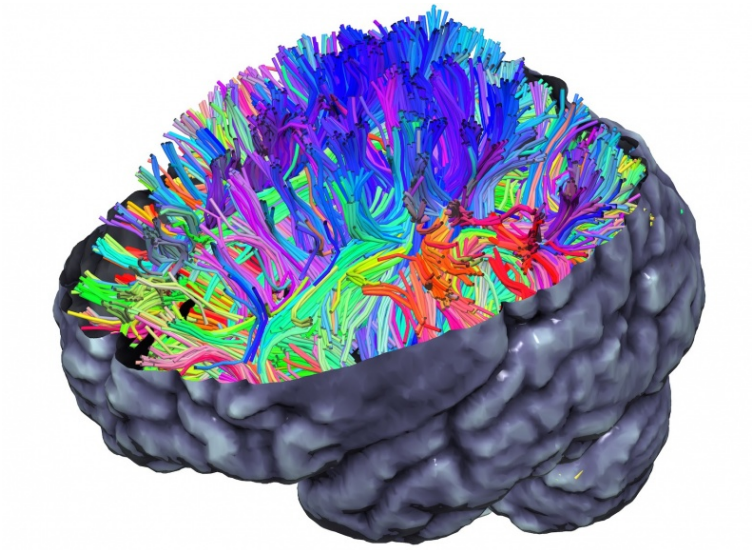


Fig. 3.23: An example tractography visualizations from **SurfIce**.

Brain Network Visualizations

Connectivity-based neuroscience research often utilizes dedicated network visualizations such as adjacency matrices (heatmaps), chord diagrams, or 3D brain network plots.

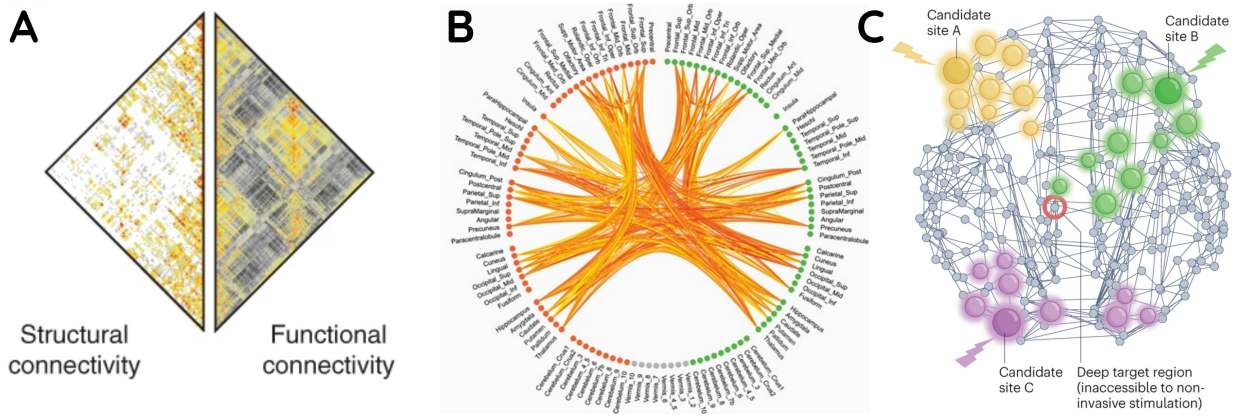


Fig. 3.24: Different visualizations of brain connectivity information via (A) heatmaps, adapted from Zamani Esfahlani *et al.*[6], (B) chord diagrams, adapted from Klauser *et al.*[7], and (C) network plots, adapted from Seguin *et al.*[8].

 Python tools for brain connectivity visualization:

The following software packages can be used to produce these maps:

- Heatmaps:
  - General purpose libraries like Matplotlib, Seaborn, and Plotly can be used to programmatically generate connectivity matrix heatmaps.
  - **Nilearn** contains functions to automate this process.
- Chord diagrams:
  - Specific packages such as **pyCircos**, **Chord**, and **OpenChord** were specifically built to make chord diagrams.
  - The **MNE** tools library also has dedicated a section on chord diagrams for connectivity.
- Brain Network visualizations:
  - **Cerebro Brain Viewer**
  - **SurfIce**

[6] Farnaz Zamani Esfahlani, Joshua Faskowitz, Jonah Slack, Bratislav Mišić, and Richard F Betzel. Local structure-function relationships in human brain networks across the lifespan. *Nature communications*, 13(1):2053, 2022.

[7] Paul Klauser, Vanessa L Cropley, Philipp S Baumann, Jinglei Lv, Pascal Steullet, Daniella Dwir, Yasser Alemán-Gómez, Meritxell Bach Cuadra, Michel Cuenod, Kim Q Do, and others. White matter alterations between brain network hubs underlie processing speed impairment in patients with schizophrenia. *Schizophrenia Bulletin Open*, 2(1):sgab033, 2021.

[8] Caio Seguin, Olaf Sporns, and Andrew Zalesky. Brain network communication: concepts, models and applications. *Nature reviews neuroscience*, 24(9):557–574, 2023.

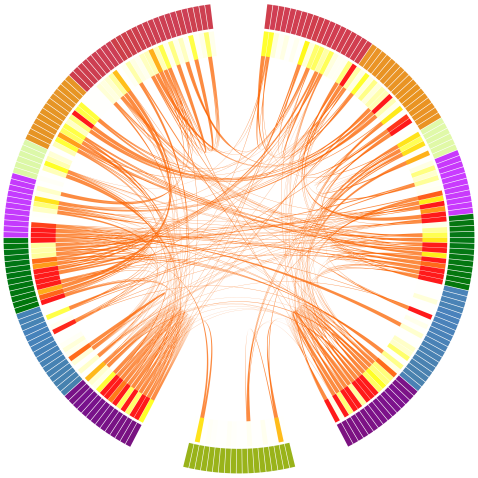


Fig. 3.25: Example chord diagram made by **pyCircos**.

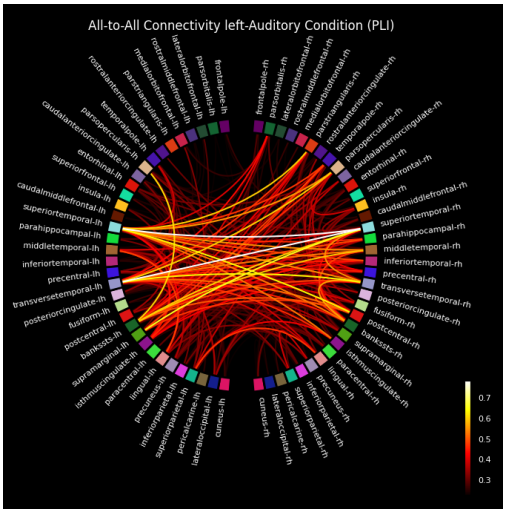


Fig. 3.26: An example chord diagram visualizations from **MNE Connectivity**.

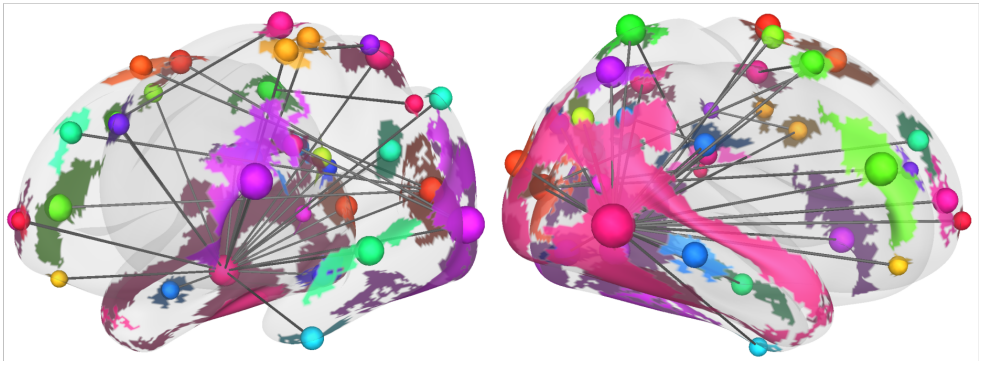


Fig. 3.27: A 3D brain network visualization from **Cerebro Brain Viewer**.

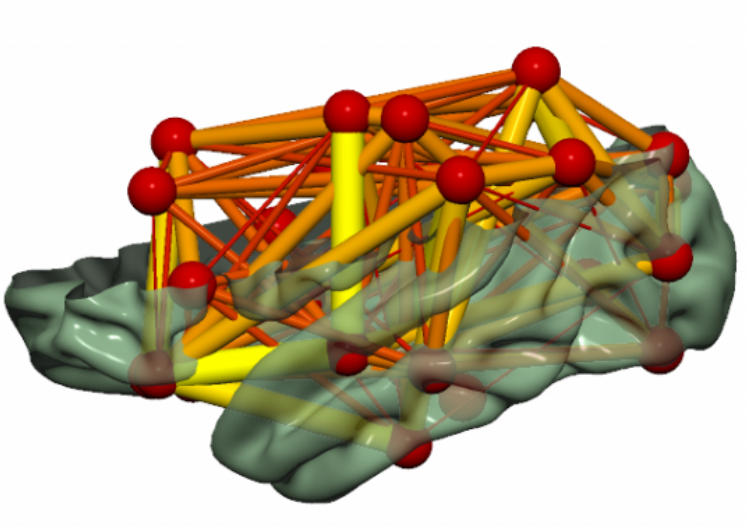


Fig. 3.28: An example brain network visualizations from **SurfIce**.

More Complex Visualizations

While beyond the scope of this 30-minute educational session, it should be noted that you could leverage full-fledged 3D rendering engines to create cinematic high-quality visuals and animations from neuroimaging data. For instance, you could use Python to drive visualizations in Blender, a powerful open-source graphics suite.

For example, here is an animation created using a Python-Blender script:

The script to reproduce this figure is available here, provided for individuals interested in further diving down this rabbit hole!

3.2.3 Supplemental Guides and Resources

The neuroimaging community is continuously developing and curating exhaustive lists of visualization tools across multiple programming languages. As promised, below are a few recommended resources to explore further. These include methodological papers, curated galleries, and practical tools to help you go beyond the examples provided in this chapter.

Key Publications

- Pernet and Madan[9]'s "Data visualization for inference in tomographic brain imaging" provides a structured guide to visualization choices for brain imaging, with helpful discussions on colormap selection and interpretability.
- Chopra *et al.*[10]'s "A Practical Guide for Generating Reproducible and Programmatic Neuroimaging Visualizations" presents a cross-language (R, Python, MATLAB) survey of tools with a focus on reproducibility and best practices.
- Chamberland *et al.*[11]'s "Tractography visualization" (Chapter in the Handbook of Diffusion MR Tractography) offers an in-depth look at diffusion imaging and fiber tracking visualization tools.

[9] Cyril R Pernet and Christopher R Madan. Data visualization for inference in tomographic brain imaging. *European Journal of Neuroscience*, 2019.

[10] Sidhant Chopra, Loïc Labache, Elvisha Dhamala, Edwina R Orchard, and Avram Holmes. A practical guide for generating reproducible and programmatic neuroimaging visualizations. *Aperture Neuro*, 3:1–20, 2023.

[11] Maxime Chamberland, Charles Poirier, Tom Hendriks, Dmitri Shastin, Anna Vilanova, and Alexander Leemans. Tractography visualization. In *Handbook of Diffusion MR Tractography*, pages 381–393. Elsevier, 2025.

Tools and Curated Resources

- **Python Graph Gallery**: A comprehensive collection of general-purpose visualization examples built with matplotlib, seaborn, plotly, and more.
- **DataCamp's Data Visualization Cheat Sheet**: A tutorial on most common general purpose visualizations and where to use them.
- **BrainCode** A code template generator for programmatic brain visualizations in R and Python. Great for learning syntax.
- **NeuroHackAcademy's Data Visualization in Python** Lecture is also worth checking out.

3.2.4 References

3.3 3 Practical Visualization Examples

This notebook presents hands-on examples that build on the concepts introduced in earlier chapters. It is intended as a practical reference, offering reusable code snippets and templates that you can adapt for your own projects. Use this resource to deepen your understanding and to develop reproducible, high-quality visualizations.

3.3.1 Notebook Preparations

Before diving into the visualization scripts, run the following (collapsed) setup cells to:

- Connect to Google Colab (if applicable)
- Import all necessary packages
- Configure the required directory structure
- Learn about the data used in these examples

Once these steps are complete, you'll be ready to begin running the visualization workflows.

Google Colab

This chapter is designed to be fully interactive and can be run directly in a Google Colab environment. This allows you to experiment with the provided scripts, modify parameters, and explore how different choices affect the resulting visualizations.

To open this notebook in Colab, use the link below:

If you're running this notebook in Colab, be sure to execute the cell below to install all required dependencies.

```
%%bash

# clone the repository
git clone https://github.com/sina-mansour/ohbm2025-reproducible-research.git

# install requirements
cd "ohbm2025-reproducible-research"
pip install -r requirements.txt
```

```
import os

# change to notebook directory
os.chdir("ohbm2025-reproducible-research/ohbm2025-reproducible-research/chapters/03/")
```

Package Imports

The cell below imports all the packages required to run this notebook. It assumes that the necessary dependencies listed in `requirements.txt` have already been installed.

```
import os
import sys
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy import sparse
import colorcet as cc
import contextlib
from io import StringIO
import nibabel as nib
import json

# Cerebro brain viewer
from cerebro import cerebro_brain_utils as cbu
from cerebro import cerebro_brain_viewer as cbv
```

Directory Setup

Next, let's set up the directory structure that will be used throughout this tutorial. In this case, we'll simply change into the appropriate directory, as the required data is already included as part of this repository.

```
# Initialize the working directory
working_directory = os.getcwd()
# create a flag to indicate whether the directory setup is complete
if 'directory_setup_complete' not in globals():
    directory_setup_complete = False
# only proceed if the directory setup is not complete
if not directory_setup_complete:
    # change the working directory to the home directory of the repository
    os.chdir("../../..")
    # print the current working directory to make sure it is correct
    working_directory = os.getcwd()
    print(f"Current working directory: {working_directory}")
    # set the flag to indicate that the directory setup is complete
    directory_setup_complete = True
else:
    # print a message indicating that the directory setup is already complete
    print("Directory setup is already complete. No changes made.")
    print(f"Current working directory: {working_directory}")
```

```
Current working directory: /mnt/local_storage/Research/Codes/jupyterbooks/ohbm2025-
↪reproducible-research
```

Tutorial Data

A minimal set of neuroimaging files has been prepared to support the execution of the examples in this notebook.

Note: This curated data is provided exclusively for educational purposes as part of this Jupyter Book tutorial.

For access to complete datasets or for use beyond this tutorial, please refer to the original data sources listed below.

Data Sources

The neuroimaging data used in this tutorial are derived from the following publicly available resources:

- **Human Connectome Project**'s Group Average Adult Template (see Glasser *et al.*[1], Van Essen *et al.*[2], and Marcus *et al.*[3])
- The Glasser Cortical brain atlas (see Glasser *et al.*[4])
- Tractography data from the ORG fiber clustering atlas (see Zhang *et al.*[5])
- Functional connectivity data from Mansour *et al.*[6] (more information)
- Melbourne subcortical atlas (see Tian *et al.*[7])

3.3.2 Reproducible NeuroImaging Visualizations

With the directory structure in place and example neuroimaging data made available, we can now turn our attention to creating a variety of code-based visualizations.

These examples are designed to illustrate best practices for reproducible neuroimaging visualization workflows and demonstrate how to effectively explore and present brain imaging data using code.

Volume Slice Rendering

In this section, we'll demonstrate how to render volume slices using data from the Human Connectome Project's **S1200 Group Average Data Release**. Specifically, we'll visualize a set of axial slices from the group-average T1-weighted image, followed by an overlay of the T2-weighted image using an arbitrary intensity threshold.

The images will be loaded using **Nibabel**, and slices will be visualized using **Matplotlib**. To promote reusability and reproducibility, we'll structure the code as modular, well-documented functions.

```
visualize_multiple_axial_slices(f"{working_directory}/data/S1200_AverageT1w_restore.
↪nii.gz")
```

[1] Matthew F Glasser, Stamatios N Sotiropoulos, J Anthony Wilson, Timothy S Coalson, Bruce Fischl, Jesper L Andersson, Junqian Xu, Saad Jbabdi, Matthew Webster, Jonathan R Polimeni, and others. The minimal preprocessing pipelines for the human connectome project. *Neuroimage*, 80:105–124, 2013.

[2] David C Van Essen, Kamil Ugurbil, Edward Auerbach, Deanna Barch, Timothy EJ Behrens, Richard Bucholz, Acer Chang, Liyong Chen, Maurizio Corbetta, Sandra W Curtiss, and others. The human connectome project: a data acquisition perspective. *Neuroimage*, 62(4):2222–2231, 2012.

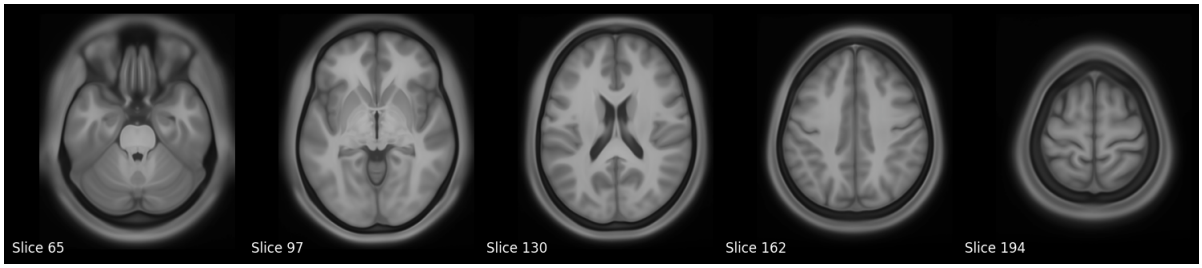
[3] Daniel S Marcus, Michael P Harms, Abraham Z Snyder, Mark Jenkinson, J Anthony Wilson, Matthew F Glasser, Deanna M Barch, Kevin A Archie, Gregory C Burgess, Mohana Ramaratnam, and others. Human connectome project informatics: quality control, database services, and data visualization. *Neuroimage*, 80:202–219, 2013.

[4] Matthew F Glasser, Timothy S Coalson, Emma C Robinson, Carl D Hacker, John Harwell, Essa Yacoub, Kamil Ugurbil, Jesper Andersson, Christian F Beckmann, Mark Jenkinson, and others. A multi-modal parcellation of human cerebral cortex. *Nature*, 536(7615):171–178, 2016.

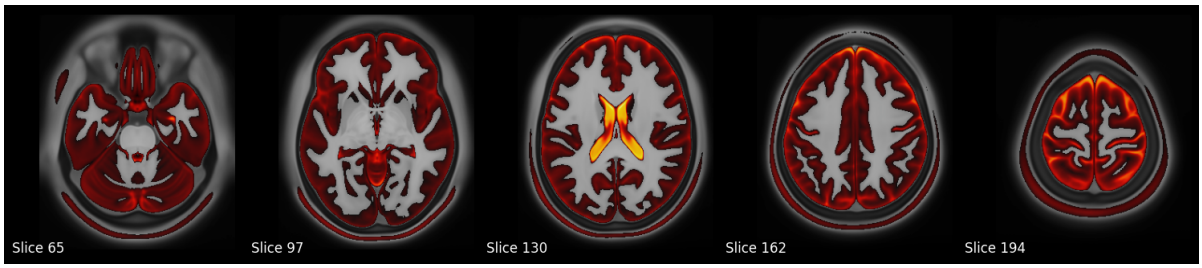
[5] Fan Zhang, Ye Wu, Isaiah Norton, Laura Rigolo, Yogesh Rathi, Nikos Makris, and Lauren J O'Donnell. An anatomically curated fiber clustering white matter atlas for consistent white matter tract parcellation across the lifespan. *Neuroimage*, 179:429–447, 2018.

[6] Sina Mansour, Ye Tian, BT Thomas Yeo, Vanessa Cropley, Andrew Zalesky, and others. High-resolution connectomic fingerprints: mapping neural identity and behavior. *NeuroImage*, 229:117695, 2021.

[7] Ye Tian, Daniel S Margulies, Michael Breakspear, and Andrew Zalesky. Topographic organization of the human subcortex unveiled with functional connectivity gradients. *Nature neuroscience*, 23(11):1421–1432, 2020.




```
# Visualize T1 weighted image, and overlay T2w on top of it
visualize_multiple_axial_slices_with_overlay(
    f"{working_directory}/data/S1200_AverageT1w_restore.nii.gz",
    f"{working_directory}/data/S1200_AverageT2w_restore.nii.gz"
)
```

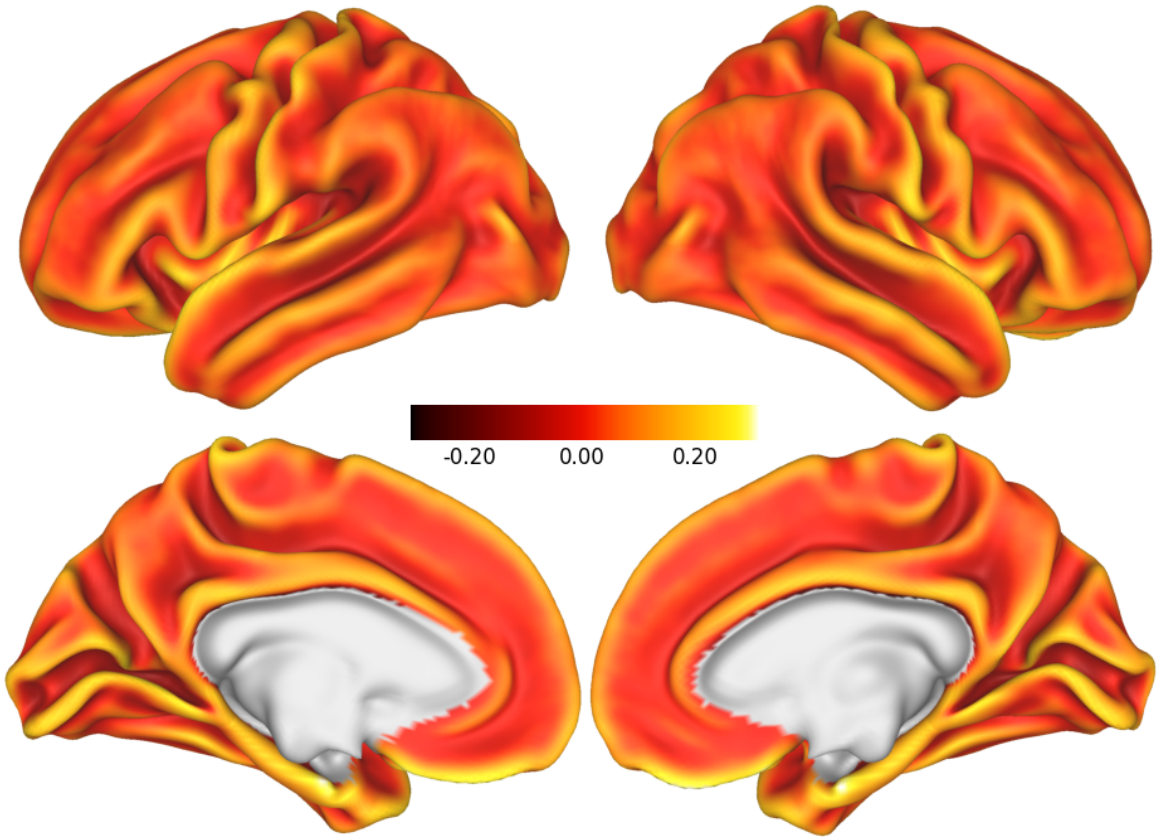


Surface-based Visualizations

In the examples below, we will work with surface-based data from the Human Connectome Project's **S1200 Group Average Data Release**, specifically using the fsLR template surface coordinates.

We'll visualize the average cortical curvature mapped onto the cortical surface to highlight anatomical landmarks and folding patterns.

```
# Plot curvature dscalar file in gray scale
dscalar_file = f"{working_directory}/data/S1200.curvature_MSMAll.32k_fs_LR.dscalar.nii
↪"

# Create a figure and axis for the plot
fig, ax = plt.subplots(figsize=(12, 9))

# Plot the dscalar file with Cerebro
plot_dscalar_with_cerebro(dscalar_file, fig=fig, ax=ax, colormap=cc.cm.fire, show_
↪colorbar=True, colorbar_format='%.2f')
```



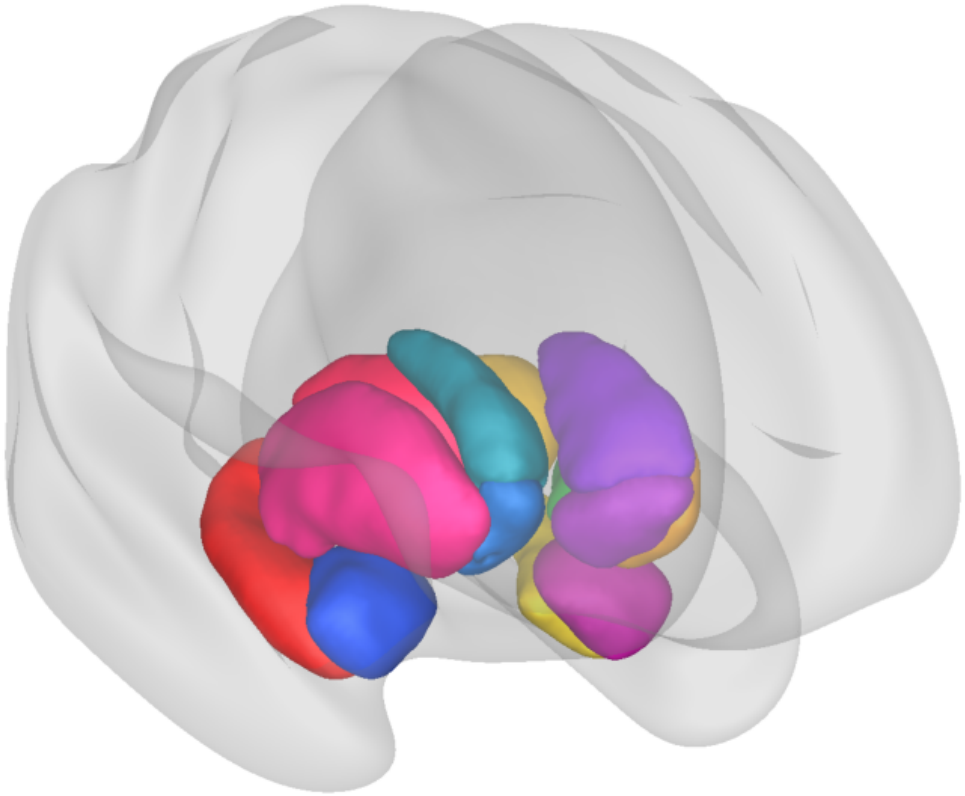
Volume-to-Surface Transformation

In this example, we demonstrate how to perform a volume-to-surface transformation using the Melbourne Subcortical Atlas, specifically from its Scale 1 parcellation (available from `Tian_Subcortex_S1_3T.nii.gz`).

We will generate and render a separate cortical surface visualization for each labeled region in the atlas.

This transformation and rendering will be performed using tools from the **Cerebro Brain Viewer**, which provides convenient functionality for mapping volumetric data onto surface meshes.

```
# Create a figure and axis for the plot
fig, ax = plt.subplots(figsize=(10, 9))

# Suppress low-level output
old_stdout, old_stderr = suppress_c_output()

try:
    # Plot the glass brain with the subcortical atlas
    plot_glass_brain_and_atlas_with_cerebro(
        ax=ax, atlas_file="data/Tian_Subcortex_S1_3T.nii.gz", view=((250, 350, 0),␣
↪None, None, None), surface='inflated', glass_color=(0.9, 0.9, 0.9, 0.2)
    )
finally:
    # Restore the original stdout and stderr
    restore_c_output(old_stdout, old_stderr)
```



Tractography Visualization

In this section, we will visualize tractography data from the ORG fiber clustering atlas. We will render a single fiber bundle using the Cerebro Brain Viewer, showcasing how to load and display streamline data effectively.

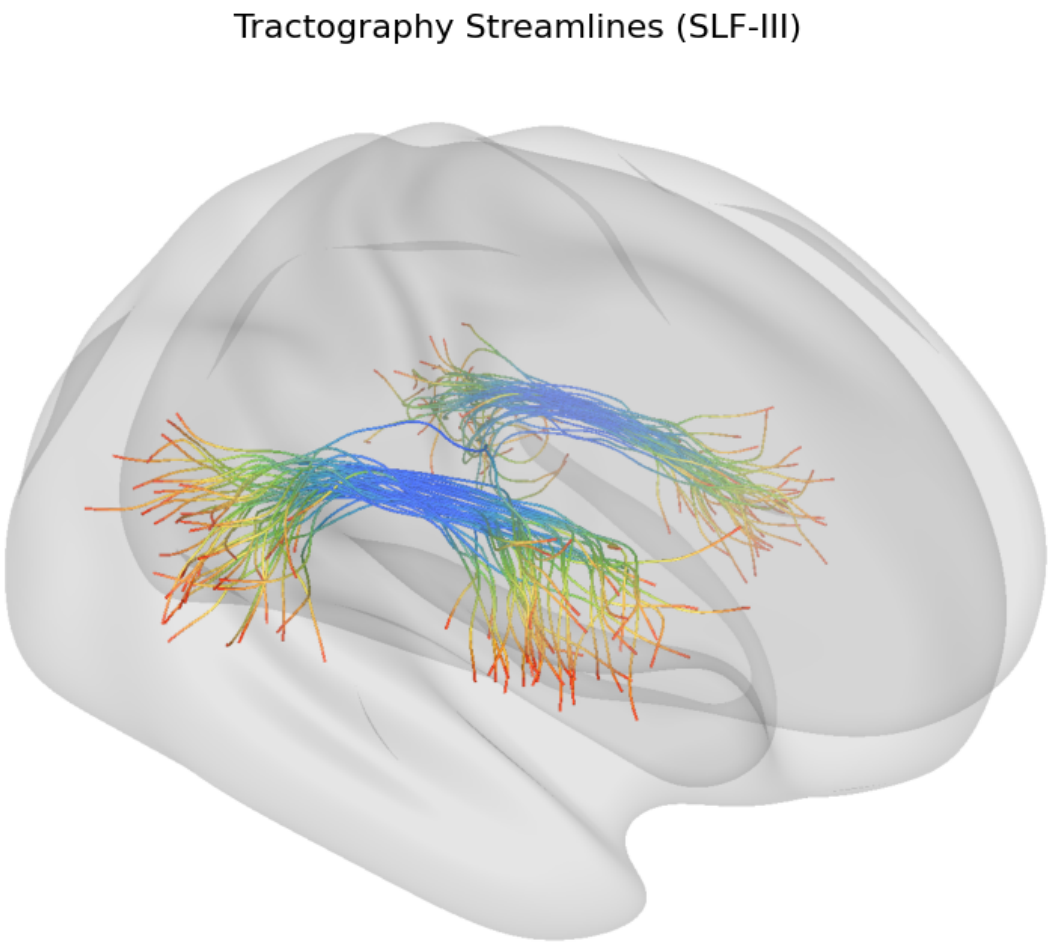
```
# Create a figure and axis for the plot
fig, ax = plt.subplots(figsize=(10, 9))

# Suppress low-level output
old_stdout, old_stderr = suppress_c_output()

try:
    # Plot the glass brain with the subcortical atlas
    plot_glass_brain_and_tractography_with_cerebro(
        ax=ax, # axis to render the brain view on
        tract_file="data/T_SLF-III-cluster_00209.tck", # path to the tractography file
        view=((400, 150, 100), None, None, None), # camera view configuration for the␣
↪brain viewer (a view from the right side, slightly tilted towards the frontal␣
↪superior part of the brain)
```

(continues on next page)

(continued from previous page)

```
        surface='inflated', glass_color=(0.9, 0.9, 0.9, 0.2)
    )
finally:
    # Restore the original stdout and stderr
    restore_c_output(old_stdout, old_stderr)

# Add text to the plot
ax.text(0.5, 0.95, "Tractography Streamlines (SLF-III)", transform=ax.transAxes,␣
↪fontsize=16, ha='center', va='center', color='black')
```

```
Text(0.5, 0.95, 'Tractography Streamlines (SLF-III)')
```



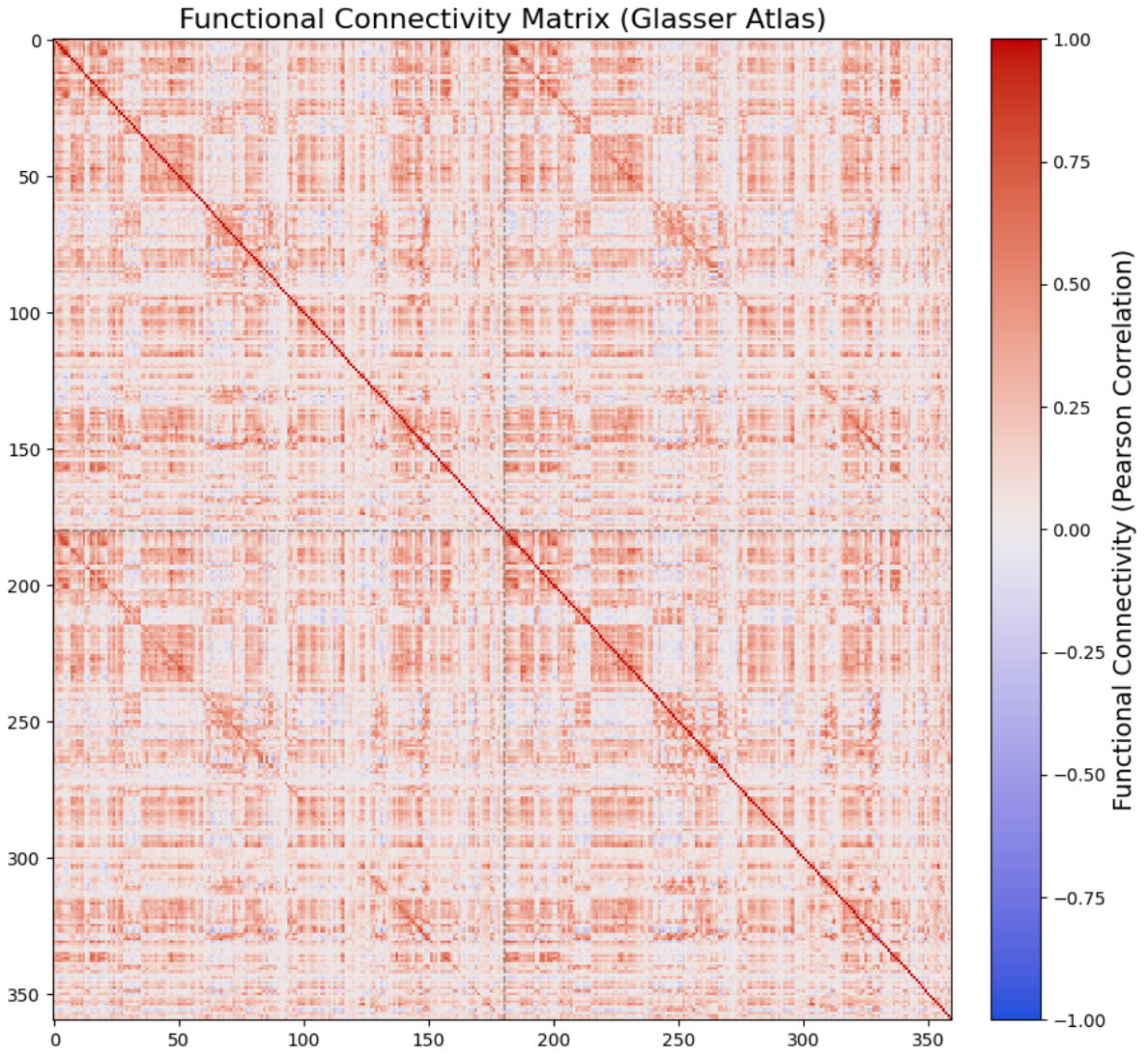
Brain Network Visualizations

This example uses functional human connectome data from Mansour *et al.*[Page 35, 6]. We will visualize both a connectivity heatmap and a 3D brain network derived from an individual's functional connectome, mapped onto the HCP MMP1 atlas (Glasser *et al.*[Page 35, 4]).

The connectivity heatmap will be created using **Matplotlib**, while the 3D brain network visualization will be generated with the **Cerebro Brain Viewer**.

```
# Get a figure and axis for the heatmap
fig, ax = plt.subplots(figsize=(10, 10))

# Plot the heatmap of the functional connectivity matrix
im = ax.imshow(fc_matrix_reordered, cmap=cc.cm.coolwarm, aspect='auto', interpolation=
 ↪'nearest', vmin=-1, vmax=1)

# Set the ticks and labels for the heatmap
glasser_label_names = [glasser_atlas_labels_dict[x][0] for x in range(1, max(glasser_
 ↪atlas_labels_dict.keys()) + 1)]

# Add a dashed line at the middle of the heatmap to separate left and right␣
 ↪hemispheres
ax.axvline(x=len(glasser_label_names)//2, color='gray', linestyle='--', linewidth=1)
ax.axhline(y=len(glasser_label_names)//2, color='gray', linestyle='--', linewidth=1)

# Add a colorbar to the heatmap
cbar = fig.colorbar(im, ax=ax, orientation='vertical', fraction=0.05, pad=0.04)
cbar.set_label('Functional Connectivity (Pearson Correlation)', fontsize=14)

# Set the title for the heatmap
ax.set_title("Functional Connectivity Matrix (Glasser Atlas)", fontsize=16)
```

```
Text(0.5, 1.0, 'Functional Connectivity Matrix (Glasser Atlas)')
```



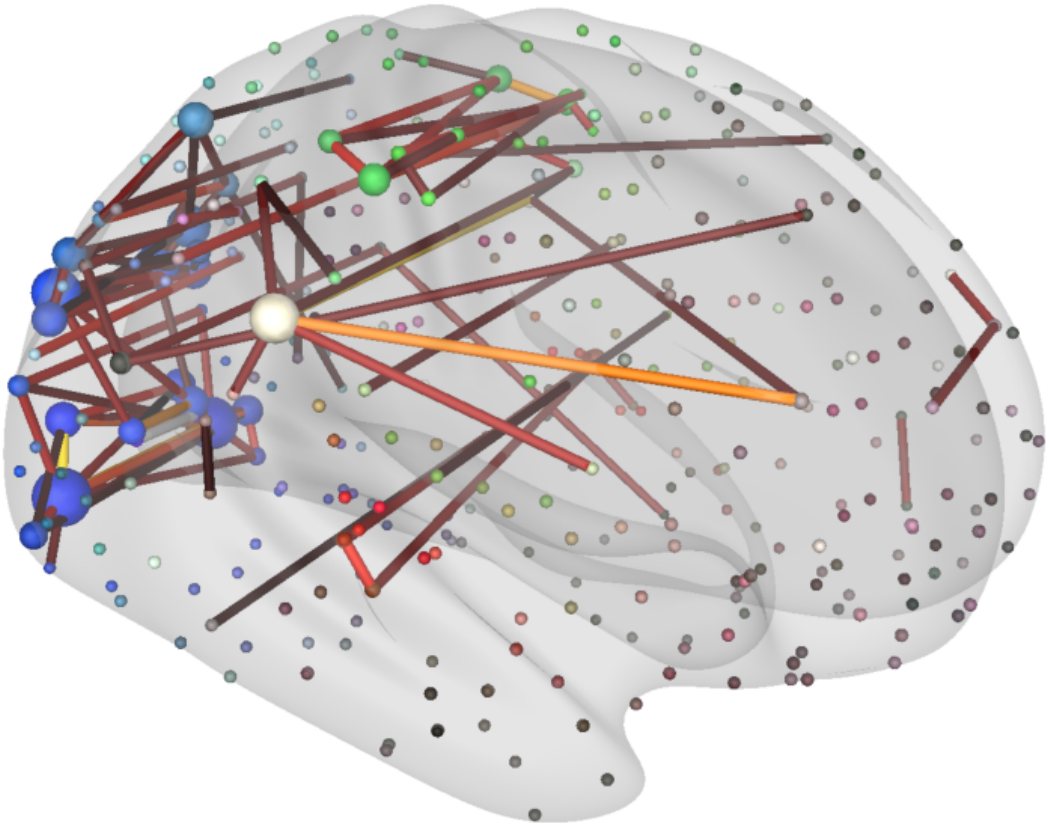

```
# Now that we have the functional connectivity matrix,
# we need the coordinates of the nodes in the glasser atlas
node_coords = np.array([lrxyz[surface_mask][glasser_atlas.get_fdata()[0] == x].
 ↪mean(axis=0) for x in range(1, max(glasser_atlas_labels_dict.keys())+1)])

# Let's also extract node colors from the glasser atlas
glasser_label_colors = [glasser_atlas_labels_dict[x][1] for x in range(1, max(glasser_
 ↪atlas_labels_dict.keys()) + 1)]

# Now let's threshold the functional connectivity matrix to only keep the strongest␣
 ↪connections
threshold = 0.7  # threshold for the functional connectivity matrix
fc_matrix_thresholded = np.where(np.abs(fc_matrix_reordered) > threshold, fc_matrix_
 ↪reordered, 0)
# also remove self-connections
np.fill_diagonal(fc_matrix_thresholded, 0)

# Node sizes can be set based on the degree of each node
node_degrees = np.sum(np.abs(fc_matrix_thresholded), axis=1)
```

(continues on next page)

(continued from previous page)

```
node_radii = np.clip(node_degrees / np.max(node_degrees) * 5, 1, 5)  # Scale node␣
↪sizes between 1 and 5
node_radii = np.repeat(node_radii[:, np.newaxis], 3, axis=1) # Make node radii along␣
↪3 dimensions (N x 3)

# Edge radii can be set based on the strength of the connections
edge_radii = np.clip(np.abs(fc_matrix_thresholded) / np.max(np.abs(fc_matrix_
↪thresholded)), 0.2, 1)  # Scale edge sizes between 0.2 and 1

# Create a figure and axis for the plot
fig, ax = plt.subplots(figsize=(10, 9))

# Suppress low-level output
old_stdout, old_stderr = suppress_c_output()

try:
    # Plot the glass brain with the subcortical atlas
    plot_glass_brain_and_network_with_cerebro(
        ax=ax, # axis to render the brain view on
        adjacency_matrix=fc_matrix_thresholded,  # The adjacency matrix of the network
        node_coords=node_coords,  # The coordinates of the nodes in the network
        node_colors=glasser_label_colors,  # The colors of the nodes in the network
        node_radii=node_radii,  # The radii of the nodes in the network
        edge_radii=edge_radii,  # The radii of the edges in the network
        view=((400, 150, 100), None, None, None), # camera view configuration for the␣
↪brain viewer (a view from the right side, slightly tilted towards the frontal␣
↪superior part of the brain)
        surface='inflated', glass_color=(0.9, 0.9, 0.9, 0.2)
    )
finally:
    # Restore the original stdout and stderr
    restore_c_output(old_stdout, old_stderr)
```

```
(206,) (206, 2) (206, 4)
```



3.3.3 ? References

CHAPTER

FOUR

REFERENCES

BIBLIOGRAPHY

[1] Marcus R Munafò, Brian A Nosek, Dorothy VM Bishop, Katherine S Button, Christopher D Chambers, Nathalie Percie du Sert, Uri Simonsohn, Eric-Jan Wagenmakers, Jennifer J Ware, and John PA Ioannidis. A manifesto for reproducible science. *Nature human behaviour*, 1(1):0021, 2017.

[2] Krzysztof J Gorgolewski and Russell A Poldrack. A practical guide for improving transparency and reproducibility in neuroimaging research. *PLoS biology*, 14(7):e1002506, 2016.

[3] Rotem Botvinik-Nezer and Tor D Wager. Reproducibility in neuroimaging analysis: challenges and solutions. *Biological Psychiatry: Cognitive Neuroscience and Neuroimaging*, 8(8):780–788, 2023.

[4] Guiomar Niso, Rotem Botvinik-Nezer, Stefan Appelhoff, Alejandro De La Vega, Oscar Esteban, Joset A Etzel, Karolina Finc, Melanie Ganz, Rémi Gau, Yaroslav O Halchenko, and others. Open and reproducible neuroimaging: from study inception to publication. *NeuroImage*, 263:119623, 2022.

[5] Greg Wilson, Jennifer Bryan, Karen Cranston, Justin Kitzes, Lex Nederbragt, and Tracy K Teal. Good enough practices in scientific computing. *PLoS computational biology*, 13(6):e1005510, 2017.

[6] Giovanni Petri, Paul Expert, Federico Turkheimer, Robin Carhart-Harris, David Nutt, Peter J Hellyer, and Francesco Vaccarino. Homological scaffolds of brain functional networks. *Journal of The Royal Society Interface*, 11(101):20140873, 2014.

[7] Taylor Bolt, Shiyu Wang, Jason S Nomi, Roni Setton, Benjamin P Gold, BT Yeo, J Jean Chen, Dante Picchioni, Jeff H Duyn, R Nathan Spreng, and others. Autonomic physiological coupling of the global fmri signal. *Nature Neuroscience*, pages 1–9, 2025.

[8] Julia M Huntenburg, Pierre-Louis Bazin, and Daniel S Margulies. Large-scale gradients in human cortical organization. *Trends in cognitive sciences*, 22(1):21–31, 2018.

[9] Daniel S Margulies, Satrajit S Ghosh, Alexandros Goulas, Marcel Falkiewicz, Julia M Huntenburg, Georg Langs, Gleb Bezgin, Simon B Eickhoff, F Xavier Castellanos, Michael Petrides, and others. Situating the default-mode network along a principal gradient of macroscale cortical organization. *Proceedings of the National Academy of Sciences*, 113(44):12574–12579, 2016.

[10] Hari McGrath, Hitten P Zaveri, Evan Collins, Tamara Jafar, Omar Chishti, Sami Obaid, Alexander Ksendzovsky, Kun Wu, Xenophon Papademetris, and Dennis D Spencer. High-resolution cortical parcellation based on conserved brain landmarks for localization of multimodal data to the nearest centimeter. *Scientific reports*, 12(1):18778, 2022.

[11] Simon R Cox, Stuart J Ritchie, Elliot M Tucker-Drob, David C Liewald, Saskia P Hagenaars, Gail Davies, Joanna M Wardlaw, Catharine R Gale, Mark E Bastin, and Ian J Deary. Ageing and brain white matter structure in 3,513 uk biobank participants. *Nature communications*, 7(1):13629, 2016.

[12] Paul Klauser, Vanessa L Cropley, Philipp S Baumann, Jinglei Lv, Pascal Steullet, Daniella Dwir, Yasser Alemán-Gómez, Meritxell Bach Cuadra, Michel Cuenod, Kim Q Do, and others. White matter alterations between brain network hubs underlie processing speed impairment in patients with schizophrenia. *Schizophrenia Bulletin Open*, 2(1):sgab033, 2021.

[13] Farnaz Zamani Esfahlani, Joshua Faskowitz, Jonah Slack, Bratislav Mišić, and Richard F Betzel. Local structure-function relationships in human brain networks across the lifespan. *Nature communications*, 13(1):2053, 2022.

[14] Caio Seguin, Olaf Sporns, and Andrew Zalesky. Brain network communication: concepts, models and applications. *Nature reviews neuroscience*, 24(9):557–574, 2023.

[15] Sidhant Chopra, Loïc Labache, Elvisha Dhamala, Edwina R Orchard, and Avram Holmes. A practical guide for generating reproducible and programmatic neuroimaging visualizations. *Aperture Neuro*, 3:1–20, 2023.

[16] Cyril R Pernet and Christopher R Madan. Data visualization for inference in tomographic brain imaging. *European Journal of Neuroscience*, 2019.

[17] Christopher Rorden. From mricro to mricron: the evolution of neuroimaging visualization tools. *Neuropsychologia*, pages 109067, 2025.

[18] Maxime Chamberland, Charles Poirier, Tom Hendriks, Dmitri Shastin, Anna Vilanova, and Alexander Leemans. Tractography visualization. In *Handbook of Diffusion MR Tractography*, pages 381–393. Elsevier, 2025.

[19] Sina Mansour, Ye Tian, BT Thomas Yeo, Vanessa Cropley, Andrew Zalesky, and others. High-resolution connectomic fingerprints: mapping neural identity and behavior. *NeuroImage*, 229:117695, 2021.

[20] Matthew F Glasser, Timothy S Coalson, Emma C Robinson, Carl D Hacker, John Harwell, Essa Yacoub, Kamil Ugurbil, Jesper Andersson, Christian F Beckmann, Mark Jenkinson, and others. A multi-modal parcellation of human cerebral cortex. *Nature*, 536(7615):171–178, 2016.

[21] Matthew F Glasser, Stamatios N Sotiropoulos, J Anthony Wilson, Timothy S Coalson, Bruce Fischl, Jesper L Andersson, Junqian Xu, Saad Jbabdi, Matthew Webster, Jonathan R Polimeni, and others. The minimal preprocessing pipelines for the human connectome project. *Neuroimage*, 80:105–124, 2013.

[22] David C Van Essen, Kamil Ugurbil, Edward Auerbach, Deanna Barch, Timothy EJ Behrens, Richard Bucholz, Acer Chang, Liyong Chen, Maurizio Corbetta, Sandra W Curtiss, and others. The human connectome project: a data acquisition perspective. *Neuroimage*, 62(4):2222–2231, 2012.

[23] Daniel S Marcus, Michael P Harms, Abraham Z Snyder, Mark Jenkinson, J Anthony Wilson, Matthew F Glasser, Deanna M Barch, Kevin A Archie, Gregory C Burgess, Mohana Ramaratnam, and others. Human connectome project informatics: quality control, database services, and data visualization. *Neuroimage*, 80:202–219, 2013.

[24] Fan Zhang, Ye Wu, Isaiah Norton, Laura Rigolo, Yogesh Rathi, Nikos Makris, and Lauren J O'Donnell. An anatomically curated fiber clustering white matter atlas for consistent white matter tract parcellation across the lifespan. *Neuroimage*, 179:429–447, 2018.

[25] Ye Tian, Daniel S Margulies, Michael Breakspear, and Andrew Zalesky. Topographic organization of the human subcortex unveiled with functional connectivity gradients. *Nature neuroscience*, 23(11):1421–1432, 2020.