Aperture Neuro

Chopra S, Labache L, Dhamala E, Orchard ER, Holmes A. A Practical Guide for Generating Reproducible and Programmatic Neuroimaging Visualizations. *Aperture Neuro*. 2023;3:1-20. doi:10.52294/001c.85104

Articles describing Code

A Practical Guide for Generating Reproducible and Programmatic Neuroimaging Visualizations

Sidhant Chopra[1], Loïc Labache[1], Elvisha Dhamala[2], Edwina R Orchard[3], Avram Holmes[4]

[1] Department of Psychology, Yale University, [2] Institute of Behavioral Sciences, Feinstein Institutes for Medical Research, [3] Yale Child Study Center, Yale University, [4] Department of Psychiatry, Brain Health Institute, Rutgers University

Keywords: Neuroimaging visualization, Reproducibility, Programmatic figures, Open Science, Python, Brain Visualization

https://doi.org/10.52294/001c.85104

Aperture Neuro

Vol. 3, 2023

Neuroimaging visualizations form the centerpiece of the interpretation and communication of scientific results, and are a cornerstone for data quality control. Often, these images and figures are produced by manually changing settings on Graphical User Interfaces (GUIs). There now exist many well-documented code-based brain visualization tools that allow users to use code to programmatically generate publication-ready figures directly within programming environments such as R, Python and MATLAB. Here, we provide a rationale for the wide-spread adoption of code-generated brain visualizations by highlighting corresponding advantages in replicability, flexibility, and integration over GUI based tools. We then provide a practical guide outlining the steps required to generate these code-based brain visualizations. We also present a comprehensive table of tools currently available for programmatic brain visualizations and provide examples of visualizations and associated code as a point of reference (https://sidchop.shinyapps.io/braincode_selector/). Finally, we provide a web-app that generates simple code-templates as starting points for these visualizations (https://sidchop.shinyapps.io/braincode/).

1. INTRODUCTION

The visualization of neuroimaging data is one of the primary ways in which we evaluate data quality, interpret results, and communicate findings. These visualizations are commonly produced using graphical user interface (GUI) -based tools where individual images are opened and, within each instance, display settings are manually changed until the desired output is reached. In large part, the choice to use GUI-based software has been driven by a perception of convenience, flexibility, and accessibility. However, there now exist code-based software packages that are well-documented and do not require high-level and comprehensive knowledge of programming, making them accessible to the neuroimaging community. These tools are flexible and can generate reproducible, high-quality, and can make publication-ready brain visualizations in only a few lines of code, especially within the R, Python and MATLAB environments. Here, we first discuss the rationale for the widespread adoption of code-generated visualizations by highlighting major advantages in replicability, flexibility, and integration. We then provide a practical guide outlining the steps required to make code-based brain visualizations and provide a web-app (https://sidchop.shinyapps.io/braincode/; see Section 4) that can generate simple code-templates as starting points for these visualizations. We also present a comprehensive table of tools currently available for programmatic brain visualizations (Table 1) and provide instructive examples of visualizations and associated code as a point of reference (Figure 2-3). Finally, we outline some limitations and gaps in the current functionality of code-based tools. The focus of this guide is on human brain magnetic resonance imaging (MRI) data, but many of the principles discussed and tools provided will equally apply to visualizing data from other organs and imaging modalities such as EEG, MEG, PET and CT.

2. BENEFITS OF LEARNING TO GENERATE CODE-BASED BRAIN VISUALIZATIONS

2.1. REPLICABILITY

In recent years, there have been multiple large-scale efforts empirically demonstrating the lack of reproducibility of findings using neuroimaging data.[1] One common solution proposed for achieving robust and reliable discoveries has been to encourage scientific output that can be transparently evaluated and independently replicated. In practice, this typically entails openly sharing detailed methods, materials, code, and data. While there is a trend towards sharing code related to neuroimaging analyses, the sharing of code used to generate figures such as brain renderings and spatial maps has been relatively neglected. This gap in reproducibility is partly driven by the fact that brain figures are often created using a manual process that involves tinkering with sliders, buttons, and overlays on a GUI, concluding with a screenshot and sometimes beautification in image processing software like Illustrator, Photoshop

Table 1. Examples of code-based neuroimaging visualizations tools that can be accessed directly within R, MATLAB and Python environments.

	Voxel	Vertex	ROI	Edge	Streamlines
R					
ANTsR	+[2]	+[2]	+[2]	+[1]	
brainconn				+[1]	
brainR	+[2,6,7]		+[6,7]		
ciftitools	+[2,3]	+[2,3]	+[*,2,3]		
fsbrain	+[4]	+[3,4]	+[*,6,3,4]		
ggseg			+[1,4]		
neurobase	+[2]				
oro.nifti	+[2]				
Python					
ANTsPy	+[2]	+[2]	+[2]		
brainiak	+[2]				
Brainplotlib		+[1,11]	+[*,1,11]		
Brainspace/surfplot		+[3,4,6,7,8]	+[*,3,4,6,7,8]		
DIPY	+[2]				+[5,9]
ENIGMA TOOLBOX			+[1,3,4,6,8]		
FSLeyes	+[2,3,4]	+[2,3,4]	+[2,3,4]		+[2,5]
ggseg			+[1,4]		
graphpype				+[1,10,11]	
MMVT		+[2,4]	+[2,4]	+[1,10,11]	
MNE	+[2,4]	+[4]	+[4]		
mrivis	+[2]				
NaNSlice	+[2]				
netneurotools		+[4]	+[*,3]		
netplotbrain			+[1,2]	+[1,11]	
nilearn	+[2]	+[2,3]	+[2,3]	+[1,11]	
niwidget	+[2,3,4]	+[2,3,4]			+[5]
Pycortex	+[8,11,12]	+[4,8,11,12]	+[*,4,8,11,12]		
pySurfer		+[4]	+[*,4]		
surfice	+[2,3,4]	+[2,3,4]	+[2,3,4,6]	+[1,4,6]	+[4,5,6]
Visbrain	+[1,2,3,6]	+[1,2,3,6]	+[1,2,3,6]	+[1,2,3,6]	
MATLAB					
BrainNetViewer	+[2,3,4,6,13]		+[2,3,4,6,13]	+[1,10]	
Brainspace		+[3,4,6,7,8]	+[*,3,4,6,7,8]		
Brainstorm	+[2]	+[4,6]	+[3,4,6]		
bspmview	+[2,13]		+[2,13]		
CandlabCore	+[2,10,13]		+[2,10,13]		
ECoG/fMRI Vis toolbox		+[10]	+[*,10]		
ENIGMA TOOLBOX			+[1,10]		
FieldTrip	+[2]	+[10]			
Lead-DBS	+[2]		+[2,3,4]		
mni2fs		+[2,3]			
mrtools	+[2]	+[4,12]	+[*,2,4,12]		
plotSurfaceROIBoundary		+[4,10]	+[*,4,10]		
Vistasoft	+[2]	+[2]	+[2]		+[10]

Note: The tools listed contain functionality required to generate (at least close-to) publication-ready neuroimaging figures via user-entered code within R, MATLAB and Python environments. An interactive version of this table can be found here: https://sidchop.shinyapps.io/braincode_selector/. This list does not include cross-platform general purpose visual-

ization software. **1** = .txt/.csv (scalar, vector, matrix as input); **2** = .nii/.nii.gz (nifti as input); **3** = .cii/.gii (cifti or gifti files as input, includes any subtypes e.g. dlabel, dtseries, .surf); **4** = FreeSurfer formats as input, including .mgz, .annot, .label, .curv, .wm etc); **5** = .trk/.tck (tractograms as input); **6** = .obj (3D object format); **7** = .ply (3D polygon format); **8** = .vtk (Visualization Toolkit format); **9** = .fib (Legacy vtk format); **10** = .mat (MATLAB format); 11 = .npy/.npz (Python numpy format); **12** = .off (object file format); * = Cortex only.

or Inkscape. Such a process makes neuroimaging visualizations inherently difficult, if not impossible to replicate, even by the authors themselves.

The code used for data visualization should reflect a core feature of open science. Given that brain figures regularly form the centerpiece of interpretation within papers, conference presentations, or news reports, making sure they can be reliably regenerated is crucial for knowledge generation and dissemination. By writing and sharing code used to generate brain visualizations, a direct and tractable link is established between the underlying data and the corresponding scientific figure. While this code doesn't necessarily reflect the validity or accuracy of the scientific finding, it allows for reproducibility, instilling transparency and robustness, while demonstrating a desire to further scientific knowledge. Some even consider publishing figures that cannot be replicated as closer to advertising, rather than science.[2]

Notably, some GUI-based tools have historically offered command-line access to generate replicable visualizations (e.g., FreeView, FSLeyes, surfice), making their use potentially equally replicable to purely code-based tools. Use of these specialized command-line interfaces can provide a useful middle ground for those who have little experience with coding environments. Nonetheless, these interfaces often still have a learning curve, but can lack other advantages, such as iteration, provided by programming environment (see Sections 2.2 and 2.3). Likewise, other GUI-based tools offer replicability in the form of automatically generated batch scripts (text files containing lines of specialized commands that can be re-executed) or in-built terminals, which can be idiosyncratic and may lack documentation to make them easily usable or replicable by those not familiar with the specific software.

2.2. FLEXIBILITY AND SCALABILITY

Being able to exactly replicate a figures via code has marked advantages beyond open science practices. In particular, the ability to reprogram inputs (such as statistical maps) and settings (such as color schemes, thresholds, and visual orientations) can streamline the entire scientific workflow. Changing inputs and settings via code allows for the easy production of multiple figures, such as those resulting from multiple analyses that require similar visualizations. A simple for-loop or plotting function with altered inputs and/or settings-of-interest can be a powerful method for exploring visualization options or rapidly creating multi-panel figures. Likewise, an arduous request from a reviewer or collaborator to alter the image processing or analysis becomes less of a burden when the associated figures can be regenerated with a few lines of code, as opposed to re-pasting and re-illustrating them manually. Having a code-base with modifiable inputs can mean that the generation of visualizations requires less time, energy and effort than image and instance specific GUI-based generation. This also makes it easier to generate consistent figures across subsequent projects. Critically, the gains of writing code for figures are cumulative, and in addition to improving programming skills, one can build a code-base for figure generation that can be reused and shared throughout a scientific career.

Precise controls via code over visualization settings, such as color schemes, legend placement and camera angles, provides much greater flexibility over visualizations. Nonetheless, part of the appeal of GUI-based tools is that the presets for such settings can provide a useful starting point and reduce the decision burden on novice users. However, similar presets are often available in the form of default settings across most code-based packages, negating the need for the user to manually enter each and every choice required for creating an image. Most code-based tools also come with documentation, with R-packages on the *CRAN* or *Neuroconductor*[3] repositories requiring detailed guidance. Recent tools have started to include detailed beginner-friendly documentation in GitHub repositories, or even entire papers (e.g., Pham, Muschelli, & Mejia, 2022; Mowinckel & Vidal-Piñeiro, 2020; Huntenburg et at., 2017; Schäfer & Ecker, 2020) that provide examples of figures that can be used as starting points or templates for new users (also see Section 4). As the popularity of code-sharing for figure increases, there will be a cornucopia of templates that can be used as the basis for new figures.

While brain visualizations are often thought of as the end results of analyses, they also form a vital part of quality control for imaging data. Tools to automatically detect artefacts, de-noise the data and generate derivatives are becoming more robust, but are not yet at the stage where visualizing the data is no longer necessary. Nonetheless, when working with large datasets such as Human Connectome Project[4] or UK BioBank,[5] it is simply not feasible to use traditional GUI-based tools to visually examine the data. The time it takes to open a single file and achieve the desired visualization settings vastly compounds when working with large datasets. Knowing how to programmatically generate brain visualizations can allow for iteration of visualization code over each image of a large datasets making quality checks of each data processing step achievable. The visual outputs of each iteration can be complied into accessible documents that can be easily scrolled, with more advanced usage allowing for the creation of interactive HTML reports (see Section 3.5), similar to those created by standardized data processing tools like *fmriprep*.[6] This increased capacity to conduct visual quality control on larger datasets will improve the identification of processing errors and result in more reliable and valid findings.

2.3. INTEGRATIVE AND INTERACTIVE REPORTING

Often in neuroimaging studies, programming languages such as R, Python and MATLAB are used for statistical analysis and generating non-brain figures, but the brain fig-

ures are outsourced to separate GUI-based tools such as *FSLeyes*, *Freeview* or *ITK-snap*. Increasingly popular software such as *R Markdown*, *Quarto* and *Jupyter Notebook* allow for the mixing of prose and code in a single script, resulting in fully reproducible and publication ready papers. By using code-based tools available within the preferred environment, brain visualizations can be directly integrated and embedded within a paper or report. For instance, a fully reproducible version of the current paper can be found on GitHub. Some journals that publish neuroimaging studies are moving towards reproducible manuscripts, including reproducible figures (e.g. *eLife*, *Aperture Neuro*), with other journals like *F1000Research* and *GigaScience* even allowing on-demand re-running of code linked to the associated article via cloud-based platforms like *Code Ocean* (Code Ocean, 2021).

Neuroimaging data are often spatially 3D and can have multiple time points, adding a 4th dimension (e.g., functional imaging data). Thus, communicating findings or evaluating quality using static 2D slices is challenging, and may not be the best representation of the data, or the associated interpretations. While well-curated 3D renderings can help with spatial localisation,[7,8] in the end, static images can only provide an incomplete representation of the data, and they force researchers to choose the "best" angle or slice to show, which often involves compromising one result to emphasize another. An added advantage of some of the code-based tools is the generation of 'rich' media like interactive figures or animations, that allow users to zoom, rotate and scroll through slices. Interacting with a figure in this way can improve scientific communication of findings. Linking to or even embedding these videos or interactive figures in papers can greatly enhance the communication of findings and make papers more engaging for the reader. Such rich brain visualizations lend themselves to being shared on science communication mediums beyond academic papers—such as presentations, websites and social media—all of which can promote the communication of research with peers and reach larger audiences.[9] This last point is becoming increasingly salient as social media has become a core medium for spreading discoveries via science communication to the public,[10-12] to the research community,[13,14] and even a primary avenue for employment opportunities for early-career researchers.[15,16] Overall, public engagement is a cornerstone of science, and the images we create are at the center of the process.

3. GENERATING CODE-BASED VISUALIZATIONS

In the following sections we outline the primary steps required when generating programmatic and reproducible human brain visualizations (Figure 1), and provide tools and heuristics to guide this process.

3.1. SELECTING A PROGRAMMING LANGUAGE

The first step in generating code-based visualization can be selecting the coding language. Three of the most popular languages are R, Python and MATLAB, all of which have many options for generating brain visualizations (see Table 1). This decision can be made based on what language the user has prior experience with, or if one of these languages was used for the analysis part of a project. There can be advantages in using the same language for visualizations and analyses, as switching between separate environments, or to a GUI-based visualization software, can be a cumbersome deviation from the scientific workflow. This can make debugging errors more difficult, as the user must regularly switch programs to visually examine the results of any modifications or adjustments to prior analyses. Using the brain visualization tools that already exist within a chosen programming environment can provide instant visual feedback on the impact of modifications to processing or analysis.

Alternatively, the choice of programming language may be dictated by visualization and data-type required. For instance, visualizing streamlines from tractography may not be currently available in the R environment (Table 1), and therefore requires the use of Python or MATLAB. Other constraints may include limited access to proprietary software like MATLAB, which would necessitate the use of open-source options such as R, Python or Octave.

3.2. IDENTIFY A VISUALIZATION TYPE

Neuroimaging data and its derivatives can be visualized in multiple forms, that have different associated file-types and visualization requirements (see Section 3.3). A brief description of more popular visualization types is provided below:

Voxel. In neuroimaging, voxels are used to represent the intensity values of a 3D scan, such as an MRI or CT scan. The voxels can be rendered in different colors to indicate tissue types or other features of interest. For example, in functional MRI scans, voxels can be colored based on their level of activation, to show which areas of the brain are more active during a specific task. These visualizations are often displayed as either slices in axial, sagittal or coronal planes (Fig2A-B; Fig3A-B) or a 3D rendering of the whole brain. Statistical values are displayed as overlays on template anatomical images, that follow a common stereotaxic coordinate system (e.g., MNI152), or on individual-specific anatomical images.

Vertex. In neuroimaging, vertices are used to create a mesh representation of brain structures, such as the cerebral cortex or a subcortical regions. Each vertex has a set of coordinates that specify its location in 3D space and is connected to other vertices to form triangles, which make up the mesh. Vertices can be used to create a 3D visualization of the brain surface and color-coded based on different attributes such as sulcal depth, thickness, or functional activation. These visualizations are often displayed as 3D rendering of each hemisphere from medial and lateral views (Fig2C,G-H; Fig3C-D). Statistical values are displayed as overlays on template surfaces which follow a common stereotaxic coordinate system and fixed number of vertices (e.g., fsaverage; Fig2C,G; Fig3C-D), or on individual-specific surfaces that have been reconstructed using an anatomical image (Fig2H).

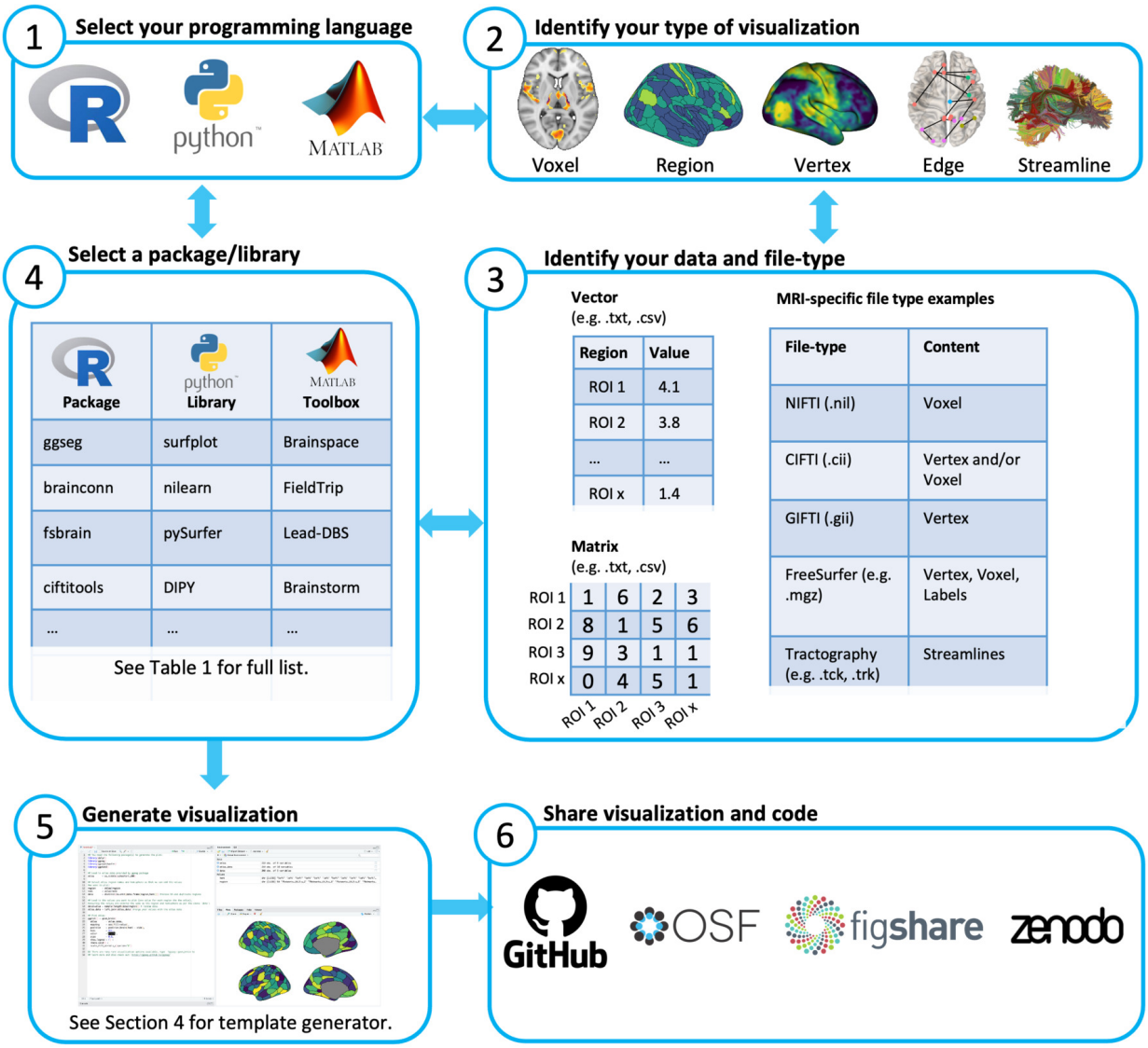

Figure 1. Primary steps to generating programmatic and reproducible human brain visualizations. Each step is outlined in Each step is outlined in the corresponding sub-section of Section 3.

Regions of Interest (ROI). In neuroimaging, ROIs are used to identify specific brain structures or areas that are relevant to the research question. ROIs can be defined using various methods, such as manual tracing, atlas-based parcellation, or functional activation patterns. Visualizing ROIs can be done by grouping and assigning the same statistical value or color to sets of voxels or vertices (Fig2B-C; Fig3B-C), or can be done through polygons. Polygonal brain visualizations are simple 2D or 3D shapes which graphically represent the brain or specific structures, but do not carry any additional information on spatial coordinates and only roughly estimate the shape of the brain and its structures. Each region of 2D (Fig 2D,F) and 3D (Fig 2E; Fig3E) polygon visualizations is filled in with colors indicating a region label or a statistical value.

Edge. In the context of neuroimaging, an edge often represents a physical or statistical connection between two brain regions, which can be visualised as a straight or curved line connecting two nodes (brain regions) in a network. One common way data is organized for edge-level visualization is a matrix, often called a 'connectivity' or 'adjacency' matrix, which indexes the presence and strength of the connections between pairs of brain regions. Visualitaion tools often convert these matrices into network graphs, where the vertices represent brain regions and the edges represent the connections between those regions, and can be displayed in a variety of ways, such as overlayed on a 2D or 3D representation of the brain (Fig2I; Fig3F), with edges represented as straight lines connecting the vertices. Often, visual properties of edges and nodes can also be adjusted to convey information, such as sized or color-coded based the strength or number of connections.

Streamlines. While similar to edges, streamlines are specifically used to represent the white matter fibers and are usually an output of conducting tractography to diffusion-weighted MRI. Streamlines are typically visualized as curved 3D lines that connect different points of the brain and can be used to create 3D visualizations of specific white

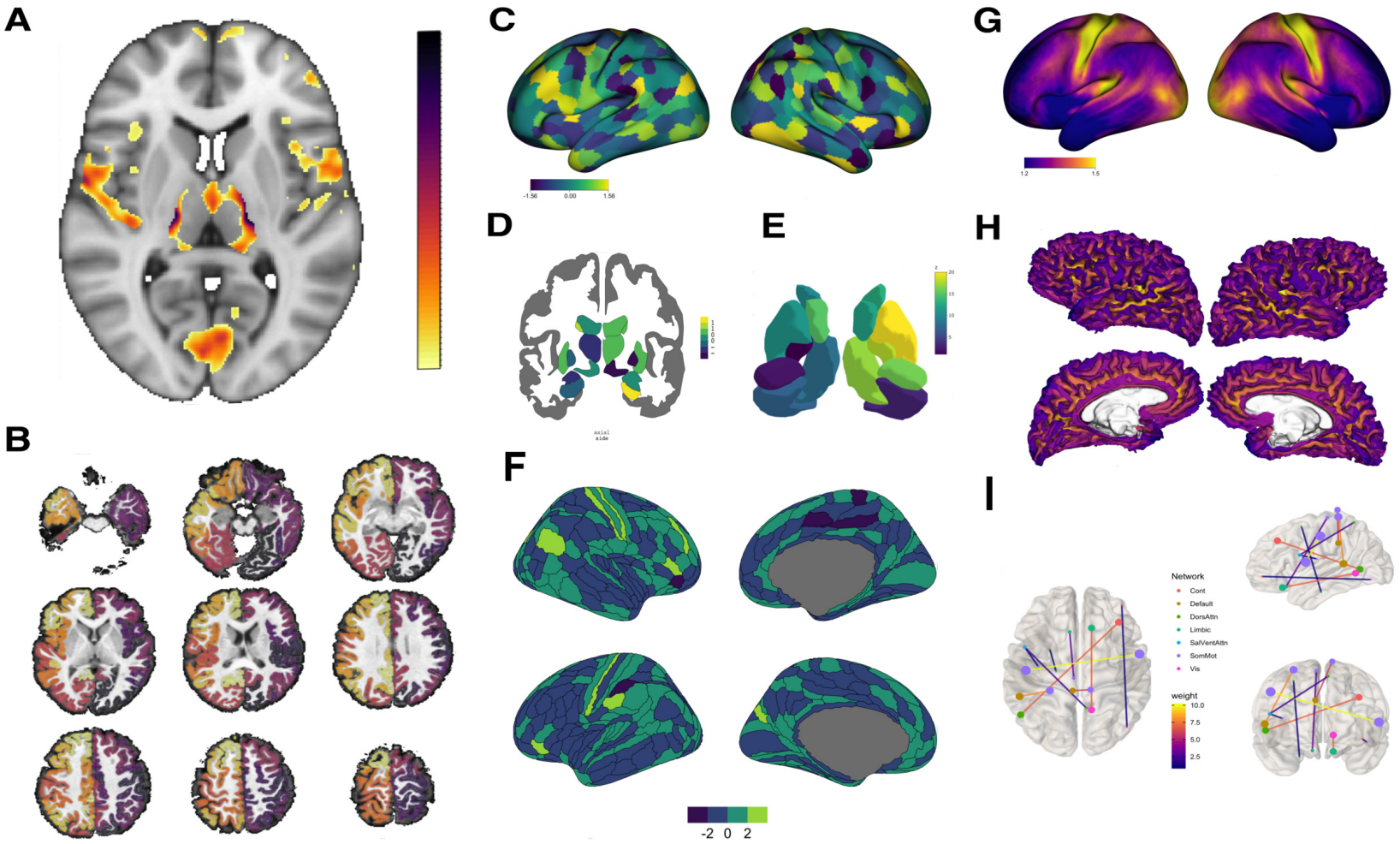

Figure 2. Examples of brain imaging visualizations made using R.

A) Voxel-level statistical map thresholded and overlaid over a T1-weighted template image, with a single axial slice shown. Made using the the `ortho2` function from the `neurobase` package. B) A voxel-level cortical parcellation overlaid on a individual T1-weighted image, shown in 9-slice axial orientation. Made using the `overlay` function from the `neurobase` package. C) A CIFTI format surface ROI atlas with a corresponding statistic assigned to each region, with both hemispheres displayed on a inflated template surface in lateral view. Made using the `view_xifti_surface` from the `ciftiTools` package. D) A coronal cross-sectional rendering of subcortical structures where a value has been assigned to each region. Made using the `aseg` atlas from the `ggseg` package. E) A 3D rendering of 9 bilateral subcortical regions where a value has been assigned to each region. Made using the `aseg` atlas from the `ggseg3d` package. F) Medial and lateral views of a ROI atlas displayed on inflated cortical surface where a value has been assigned to each region. Made using the `glasser` atlas from the `ggsegGlasser` package, which was plotted using `ggseg`. G) Lateral view of a CIFTI format vertex-level data displayed on a inflated template surface. Made using the `view_xifti_surface` function from the `ciftiTools` package. H) Medial and lateral views of vertex-level data displayed on a individuals white matter surface. Made using the `vis.subject.morph.standard` function from the `fsbrain` package. I) A weighted and undirected graph plotted on top, left and front views of a schematic outline of a brain in MNI coordinate space. Made using the `brainconn` function from the `brainconn` package. All code used to compile this figure, as well as the contents of each panel are provided in an accompanying online repository.

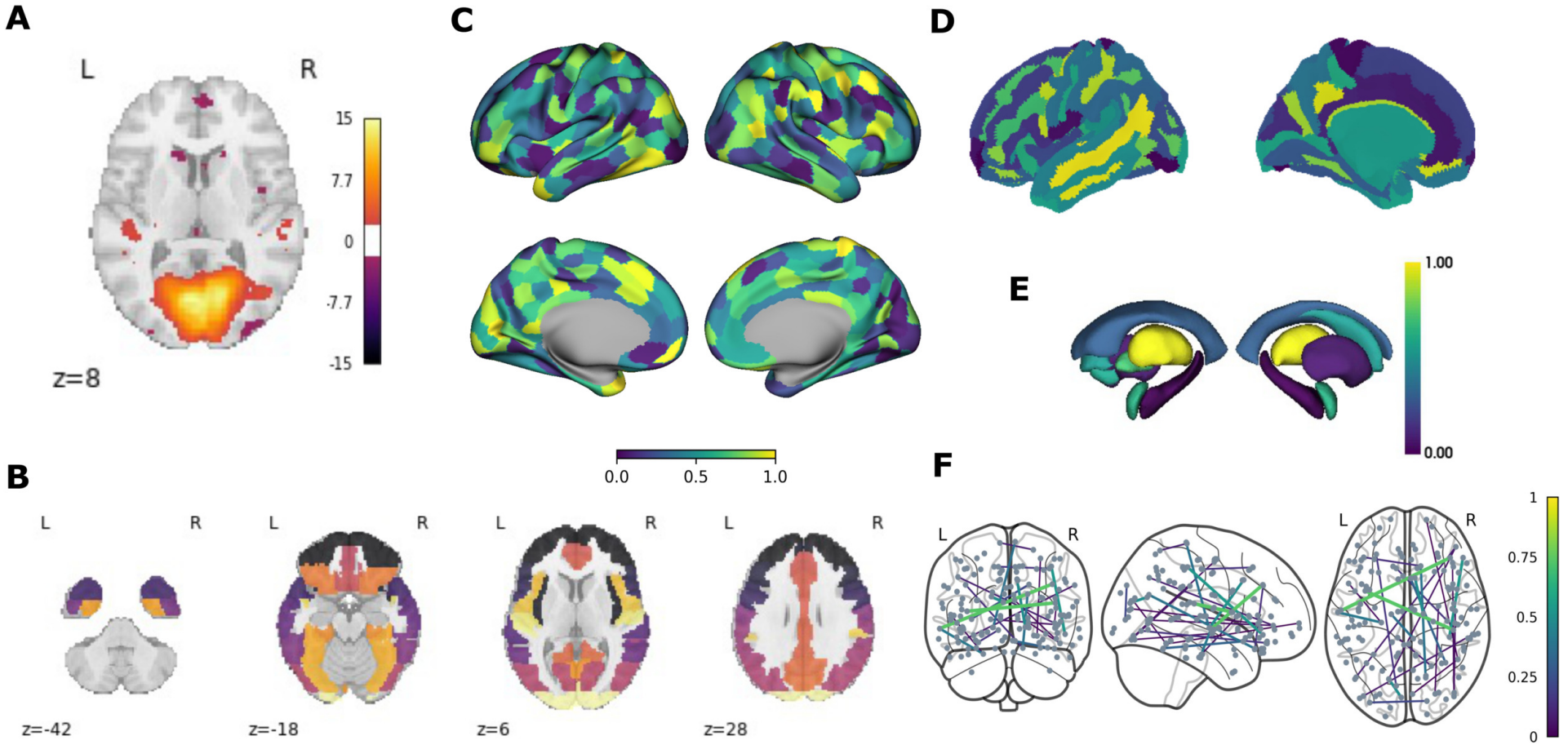

Figure 3. Examples of brain imaging visualizations made using Python.

A) Voxel-based statistical map thresholded and overlaid over a T1-weighted template data, with a single axial slice shown. Made using the `plotting.plot_stat_map` function from `nilearn`. B) Voxel-level cortical parcellation overlaid on T1-weighted MRI data shown in four axial slices. Made using `plotting.plot_roi` function from `nilearn`. C) Medial and lateral views of a cortex-wide ROI atlas, displayed on an inflated template surface where a statistical value has been randomly assigned to each ROI. Made using `Plot` function from `surfplot`. D) Lateral and medial views of vertex-level data displayed in a 3D rendering on an individual's white matter surface. Made using plotting.view_surf function in from `nilearn`. 3D rendering of 16 subcortical structures from the the Deiskan-Killiany atlas, where a statistical value has been randomly assigned to each region. Made using the `plot_subcortical` function from the `ENIGMA TOOLBOX`. A weighted and undirected graph plotted on front, right, and top views of a schematic outline of a brain in MNI coordinate space. Made using plotting.plot_connectome function in `nilearn`. All code used to generate the contents of each panel, and compile this figure are provided in an accompanying online repository.

matter tracts, or all streamlines between regions. They can be color-coded based on the direction of the fibers or properties of the tract, such as myelination. These visualizations can be overlaid on voxel- or vertex-level anatomical brain representations to provide an anatomical reference point.

The following two figures provide examples of voxel, vertex, ROI and edge-level visualizations generated within R (Figure 2) and Python (Figure 3) using open source, well documented and beginner-friendly packages. These are not an exhaustive representation of packages available for visualizing brain data in R and Python (see Table 1). Rather, the figures aim to give the reader a sense of the many options available, and are an entry point to choosing the type of brain visualization needed (also see Section 4). All code used to compile the figures, as well as the contents of each panel are provided in the accompanying online repository.

3.3. IDENTIFY INPUT FILE-FORMATS

Neuroimaging data and its derivatives come in many different file formats, and code-based visualization packages have specific formats they are designed to work with. Table 1 provides a key, listing which data-type can be used as inputs for each package. Some of the most common MRI file-formats used as inputs for visualizations are briefly described below:

Plain text format. The simplest input formats are a scalar, vector, and matrix, which represent a single datapoint, one-dimensional array of data (e.g., a single column or row), and a two-dimensional array of data (e.g. multiple columns or rows), respectively. These data are often stored in plain text formats such as `.txt`, `.csv`, and `.tsv` and can be generated by neuroimaging analyses software such as SPM, FSL or FreeSurfer. These files can also contain rows and columns of region names and/or spatial coordinates. All programming languages have functions to read these plain text formats into the coding environment. These formats are often used in region-level visualizations, where groups of voxels or vertices share the same value or color (Fig2B-F; Fig3B-E), or in edge-level visualizations, where a matrix is used to identify regions are connected by an edge (Fig2F; Fig3F).

NIfTI. NIfTI (`.nii`) files store 3D or 4D image data that are often a matrix of voxel intensities or a series of 3D matrices for 4D data such as fMRI and dMRI. The image data is stored as a 3D matrix of voxel intensities, and the header contains additional information, such as the image dimensions and voxel size. Additional information such as the subjects demographics and scanner parameters can also be stored in the header in the form of metadata.

GIfTI. GIFTI (`.gii`) is an extension of the NIFTI format, and stores data in a surface-based format represented as a set of vertices, edges, and faces that define a surface mesh. The format also includes support for storing data such as curvature, thickness, and functional activity maps on the surface mesh. Often `.gii` file names will have a pre-indicator of what information the file contains, such as `.surf.gii`, which would contain only vertices, edges and faces to define a surface mesh. Or `.func.gii` files that contain data values for every vertex, which are essentially data arrays whose indices correspond to a surface file and need a corresponding surface file to know where in the brain to assign the data values. GIFTI files can also store multiple surfaces in a single file, and can include information about the topology of the surfaces, such as the number of vertices, edges, and faces. Additional metadata can also be stored in the header.

CIfTI. CIFTI (`.cii`) files can store data from both surface-based and volume-based neuroimaging analyses, combining aspects of NIFTI and GIFTI files. For surface-based data, the file contains vertex coordinates, and data values at each vertex. For volume-based data, the file contains a 3D matrix of voxel intensities. Often this datatype is used to represent the cortex as vertices, and subcortical, brain stem and cerebellar structures as voxels. CIFTI files can be divided into three main types: `.dtseries` (store time-series data, such as fMRI data), `.dtscalar` (store scalar data, such as thickness or curvature maps), and `.dtlabel` (store label data, such as parcellations of the brain). These files can also include fields with additional metadata such as surface and volume registration information.

FreeSurfer. FreeSurfer is a commonly used image processing and analysis software that comes with a variety of proprietary formats to store outputs. The primary format is `.mgh` and its compressed version `.mgz`, which like `.nii` files, genrally stores voxel-level data. Vertex-level data is stored in multiple formats such as `.pial`, `.white`, and `.inflated`. FreeSurfer also uses `.label` files to store a list of vertices and associated labels for each brain structure, and `.annot` files FreeSurfer store annotation information such as vertices, labels and color information that can be overlaid on the surface reconstruction.

Tractography. Commonly used tractography file formats include `.trk` and `.tck` developed by the TrackVis and MRtrix software packages, respectively. Both formats store coordinates for streamlines as a series of 3D points, with each point represented by its x, y, and z coordinates, as well as a tract header with the number of points in the tract, the properties of the tract (e.g., mean diffusivity, fractional anisotropy), as well as the starting and ending indices of streamlines.

Critically, many of these datatypes are interchangeable, and can be converted between each other. There are many ways to convert between format, for instance, the Connectome Workbench,[17] and related R-packages like `ciftitools`[18] allow for conversion between NIFTI, CIFTI and GIFTI formats. FreeSurfer functions such as `mri_convert`, the NiBabel[19] library in *Python* and the `fsbrain`[20] in R, all allow for conversion between the propriety formats and open-source NIFTI, CIFTI and GIFTI formats. Therefore, if a given file format to is not compatible with a visualization package, library, or toolbox, the user can convert data into the desired format. Although, it is important to visualize and validate data after converting between data-structures, to ensure the data have not been misinterpreted, and also to be aware that unintended consequences of mapping between 2D surface and 3D volume formats can arise.[21]

3.4. SELECT A PACKAGE, LIBRARY OR TOOLBOX

The previous steps of deciding a programming environment, visualization type and input data-type will help users decide which package, library or toolbox is the right choice. Table 1 provides a list of tools in R, Python and MATLAB, classified by whether they are able to generate voxel, vertex, ROI, edge or streamline based visualizations. While each tool may contain the ability to generate code-based visualizations, some tools are more beginner-friendly and better documented than others. Usually, these tools are specifically designed for brain data visualization, as opposed to tools that are designed for brain data analysis but also provide some limited visualization functionality. Some examples of well documented and beginner-friendly tools are provided via a code template generator (see Section 4).

An important consideration when selecting a tool is whether it can generate publication-ready plots. Publication-ready plots are high resolution, labelled, contain all color bars and legends, and require no additional manual image manipulation. While all tools listed in Table 1 contain some of these features, some tools enable more precise control over publication-ready features such as legend placement, color bar placement, annotations, labeling, and multi-panel figures. These tools usually produce visualization that rely on mainstream general purpose plotting engines such as `ggplot` in R and `matplotlib` in Python. This allows users to leverage many additional features to make their brain visualizations publication ready. For example, the `ggseg` and `ggsed3d` packages[22] in R generate plots compatible with the widely-used grammar of graphics (i.e., `ggplot2`) and `plotly` engines, respectively. Similarly, the `Nilearn` library in python allows plots to be generated using `matplotlib` or `plotly` engines. We note that many of the others listed in Table 1 also rely on common plotting engines to generate visualizations.

3.5. GENERATE VISUALIZATION

Popular integrated development environments (IDE) such as *RStudio*, *Visual Studio* and *Spyder*, come with the inbuilt ability to display and update figures as the code is executed. The resulting visualization can be shared or embedded in papers in multiple different ways, with differing levels of replicability (see Section 2.3) and visual quality. One common way to share visualizations is to export it as a image raster format such as `.png`, `.tiff` or `.jpeg`, where images appear as a grid of pixels and each pixel in the grid contains information about the color and intensity of that specific point in the image. These formats are resolution dependent, and will become pixelated and difficult to parse when enlarged. Whereas vector formats, such a `.svg`, `.eps` and `.pdf` can be scaled larger or smaller without losing quality. While all coding environments provide ways to export visualizations into raster formats, exporting using vector formats, while visually superior, depends on the specific tool. Generally, only visualization tools that rely on mainstream general purpose plotting engines such as `ggplot` in R and `matplotlib` in Python allow for images to be exported as true vector formats.

Increasingly popular software such as *R Markdown*, *Quarto*, *Jupyter Notebook*, and *Google Collab* can create dynamic documents that combine code, text, and visualizations in a single file. This makes it easier to document workflows and share complete analyses, enhancing both collaboration and reproducibility. Including the code used to generate figures and other results alongside well formatted text-based explanations (see Section 2.3), enables the user and others to replicate work accurately. These tools also offer a wide range of output formats for documents, including PDF, HTML, Word and LaTeX. This versatility enables the generation of polished reports, presentations, manuscripts, or even interactive dashboards, all from a single source file. Currently, while *R Markdown* is primarily associated with R and *Google Collab* with Python, Jupyter Notebook and Quarto support a broader range of programming languages, including Python, Julia, R, and others.

3.6. SHARE VISUALIZATION CODE

Images that are generated using code can then be inserted into outputs such as a manuscript and the associated code can be included in supplementary materials. While this is the simplest way to share visualizations and code, it may not be as accessible as uploading code to dedicated code-sharing and version control platforms such as GitHub or GitLab. These services are currently widely used and allow for code to maintain formatting, version control and search functions.

Alternate options are widely used general-purpose research data repositories, such as Open Science Framework, Zenodo or FigShare, that enable researchers to publish and share their datasets, software, code and other research outputs at no cost. One advantage of these platforms is that they can assign a Digital Object Identifier (DOI) number to shared materials, making them independently citable and enhancing visibility since DOIs are indexed by various repositories and search engines. Some platforms even provide usage metrics, enabling users to gather insight on how often code and materials are accessed, cited or reused by others, which can be valuable information for evaluating the impact and reach of work.

An important consideration when both writing and sharing code for figure generation is the long-term preservation of code and resistance to software collapse. While a discussion of code and software preservation is outside the scope of the current paper, readers can refer to [NO_PRINTED_FORM], and initiatives such as Software Heritage which aim to preserve and archive all the software source code available worldwide, ensuring that valuable software source code is not lost over time.

In addition to the platform used, how accessible the code is to both the user and others depends on how clearly the code is written, formatted and commented. While guidance on proper organization of the neuroimaging visualization code is beyond the scope of this guide, we point readers to other practical guides on this topic.[24,25]

While sharing code is necessary for replicability, it is often not sufficient, as the underlying data source being visualized may be needed for the code to function correctly.

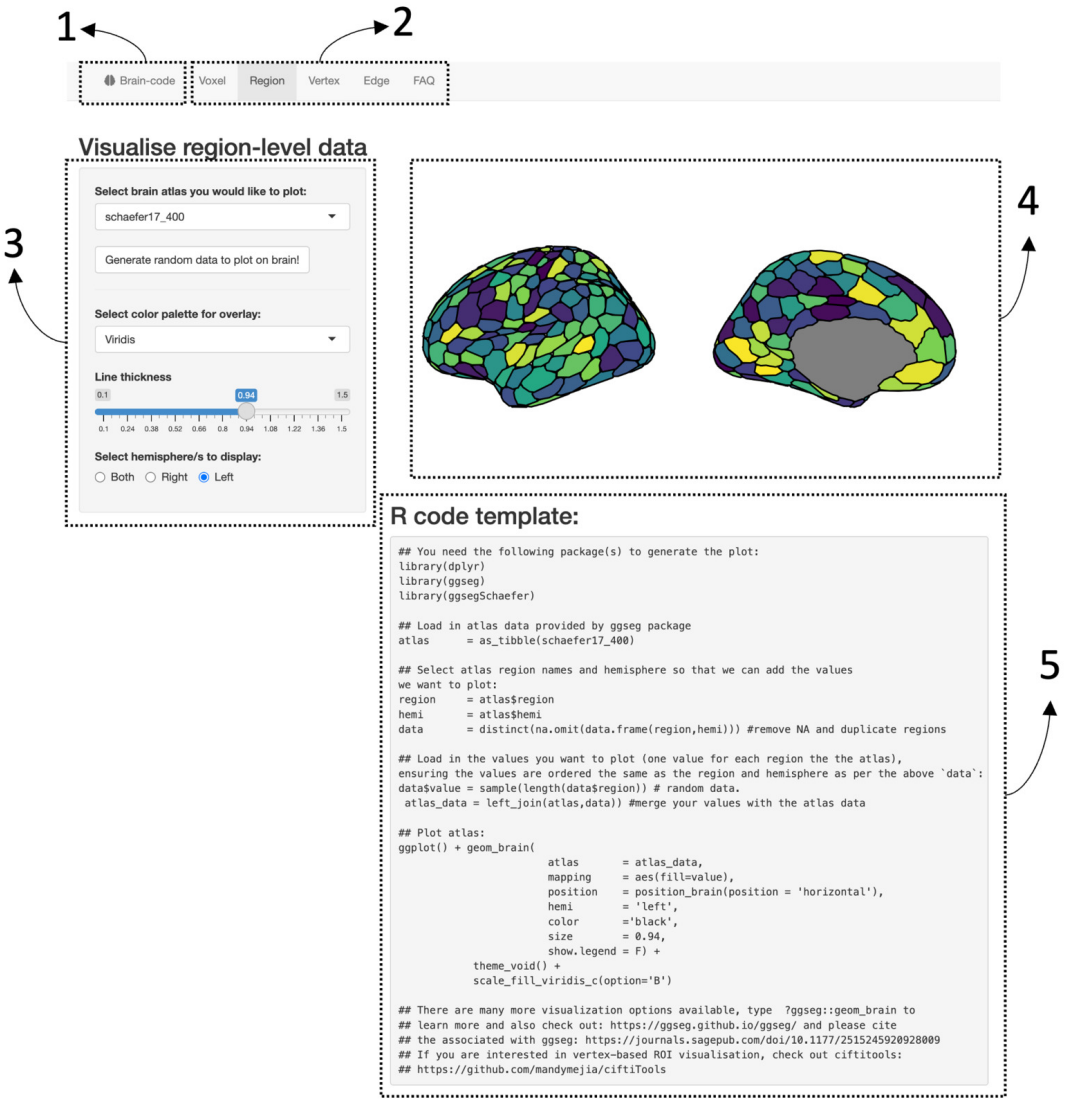

Figure 4. Interface for BrainCode web-app which generates simple code-based templates for brain visualizations.

(1) Select a programming environment (R or Python). (2) Select a visualization type (Voxel, Region, Vertex and Edge). There is also a Frequent Asked Questions tab to aid users. (3) Manually adjust limited visualization settings and examine how it reactively changes the visualization (4) and code template (5). (5) Copy the code template into selected programming environment, change file-paths to data and explore other visualization settings offered by the functions.

While many of the same sharing platforms listed above can also host source data alongside code, there now exist specialized platforms, such as OpenNeuro and NeuroVault, that allow sharing of neuroimaging specific datasets, such as those containing NIFTI and CIFTI images. If the source data for visualizations cannot be shared, synthetic data can also be generated and provided alongside the code be provided.[26]

4. BRAIN-CODE: A WEB-APP FOR GENERATING CODE TEMPLATES FOR BRAIN VISUALIZATIONS

To assist researchers transition into generating code-based brain visualizations, we have developed a web-app (https://sidchop.shinyapps.io/braincode/) that interactively generates of code-templates for beginner friendly libraries/packages in R and Python. In the web-app, users can select R or Python as their coding environment, and choose between voxel, ROI, vertex, and edge-level visualizations. They can then manually adjust a limited set of visualization setting, such as color-scales and view, and are provided with a reactive code-template that can be copied and then used within their respective programming environment. The provided code templates require users to customize the code, such as alter file-paths. The available settings have been purposefully limited to allow users to explore and fine-tune additional visualization options within their own programming environment. The code-template also contains links and prompts to more detailed documentation, alternate packages/libraries and tutorials which allow for more complex and publication-ready brain visualizations. Users can download bundled version of the web-apps via a GitHub repository.

5. LIMITATIONS AND FUNCTIONALITY GAPS

While many code-based tools are well documented and do not require a strong knowledge of programming, there can still be a steep learning curve for new users, compared to using a GUI. This is especially true for the purpose of a publication-ready figure, where fine adjustments to visual features such as legend placement, font size and multi-panel figure positioning may be needed. While most code-based tools offer some control over these finer steps, there are differences between them in feature availability and usability, with tools that use established graphic engines such as `ggplot2`, `matplotlib` and `plotly` providing the most versatile and well-documented features for visual auxiliary. Relatedly, while some interactive image viewers can be opened within an integrated development environment like R-Studio (e.g., Muschelli, 2016), for quick and interactive viewing of single GUI tools can be faster and more practical.

Often cerebellar and brain-stem regions are not well represented in software (e.g., Figure 2-3), potentially mirroring the cortico-centric sentiment that has prevailed in human neuroimaging research.[27] Likewise, custom non-cortical atlases such as non-standard subcortical atlas schemes are not yet straightforward, and usually require multiple functions and packages to visualize. Visualizing these structures often requires chaining together multiple GUI-based tools (see Madan, 2015). Alternatively users can convert neuroimaging specific file-formats into domain general visualization or polygon formats, such as `.vtk`, `.ply` or `.obj`, which can be read, manipulated and visualized using general purpose code-based tools. Examples of such tools include PyVista and Mayavi in Python and `rayshader` and `plotly` in R. Moreover, some neuroimaging derived datatypes, such as streamlines resulting from DWI-based tractography, are still not well represented in code-based visualization tools and future development should focus on enhancing visualization capabilities using these datatypes.

As can be seen in Table 1 (https://sidchop.shinyapps.io/braincode_selector/), there are usually multiple packages within each programming environment which can visualize each data type. While this provides choice for advanced users, it can also lead to confusion for novice users who may not be familiar with the nuanced differences between tools. While the process we have outlined, and the table and web-app we have provided will help users decide the ideal package, future work should continue to consolidate brain visualization methods into unified beginner-friendly code-based tools which rely on established and well-documented graphic engines and can plot multiple data types.

Submitted: November 07, 2022 CST, Accepted: July 02, 2023 CST

REFERENCES

1. Poldrack RA, Baker CI, Durnez J, et al. Scanning the horizon: Towards transparent and reproducible neuroimaging research. *Nat Rev Neurosci*. 2017;18(2):115-126. doi:10.1038/nrn.2016.167

2. Steel G. *Publishing Research without Data Is Simply Advertising, Not Science*. Open Knowledge Foundation; 2013. https://blog.okfn.org/2013/09/03/publishing-research-without-data-is-simply-advertising-not-science/

3. Muschelli J, Gherman A, Fortin JP, et al. Neuroconductor: An r platform for medical imaging analysis. *Biostatistics*. 2018;20(2):218-239. doi:10.1093/biostatistics/kxx068

4. Van Essen DC, Smith SM, Barch DM, Behrens TEJ, Yacoub E, Ugurbil K. The WU-minn human connectome project: An overview. *NeuroImage*. 2013;80:62-79. doi:10.1016/j.neuroimage.2013.05.041

5. Sudlow C, Gallacher J, Allen N, et al. UK biobank: An open access resource for identifying the causes of a wide range of complex diseases of middle and old age. *PLoS Med*. 2015;12(3):e1001779. doi:10.1371/journal.pmed.1001779

6. Esteban O, Markiewicz CJ, Blair RW, et al. fMRIPrep: A robust preprocessing pipeline for functional MRI. *Nat Methods*. 2018;16(1):111-116. doi:10.1038/s41592-018-0235-4

7. Madan CR. Creating 3D visualizations of MRI data: A brief guide. *F1000Res*. 2015;4:466. doi:10.12688/f1000research.6838.1

8. Pernet C, Madan CR. Data visualization for inference in tomographic brain imaging. Published online 2019.

9. Li Y, Xie Y. Is a picture worth a thousand words? An empirical study of image content and social media engagement. *Journal of Marketing Research*. 2019;57(1):1-19. doi:10.1177/0022243719881113

10. Mueller-Herbst JM, Xenos MA, Scheufele DA, Brossard D. Saw it on facebook: The role of social media in facilitating science issue awareness. *Social Media + Society*. 2020;6(2):205630512093041. doi:10.1177/2056305120930412

11. Smith CN, Seitz HH. Correcting misinformation about neuroscience via social media. *Science Communication*. 2019;41(6):790-819. doi:10.1177/1075547019890073

12. Huber B, Barnidge M, Gil de Zúñiga H, Liu J. Fostering public trust in science: The role of social media. *Public Underst Sci*. 2019;28(7):759-777. doi:10.1177/0963662519869097

13. Luc JGY, Archer MA, Arora RC, et al. Does tweeting improve citations? One-year results from the TSSMN prospective randomized trial. *The Annals of Thoracic Surgery*. 2021;111(1):296-300. doi:10.1016/j.athoracsur.2020.04.065

14. Quintana DS. *Twitter for Scientists*. eBook edition. Zenodo; 2020. doi:10.5281/zenodo.3707741

15. Baker M. Social media: A network boost. *Nature*. 2015;518(7538):263-265. doi:10.1038/nj7538-263a

16. Lee JSM. How to use twitter to further your research career. *Nature*. Published online February 8, 2019. doi:10.1038/d41586-019-00535-w

17. Marcus DS, Harwell J, Olsen T, et al. Informatics and data mining tools and strategies for the human connectome project. *Front Neuroinform*. 2011;5:4. doi:10.3389/fninf.2011.00004

18. Pham DD, Muschelli J, Mejia AF. ciftiTools: A package for reading, writing, visualizing, and manipulating CIFTI files in r. *NeuroImage*. 2022;250:118877. doi:10.1016/j.neuroimage.2022.118877

19. Brett M, Markiewicz CJ, Hanke M, et al. *Nipy/Nibabel: 5.0.0*. Zenodo; 2023. doi:10.5281/ZENODO.7516526

20. Schäfer T, Ecker C. fsbrain: An r package for the visualization of structural neuroimaging data. Published online September 20, 2020. doi:10.1101/2020.09.18.302935

21. Ciantar KG, Farrugia C, Galdi P, Scerri K, Xu T, Bajada CJ. Geometric effects of volume-to-surface mapping of fMRI data. *Brain Struct Funct*. 2022;227(7):2457-2464. doi:10.1007/s00429-022-02536-4

22. Mowinckel AM, Vidal-Piñeiro D. Visualization of Brain Statistics With R Packages *ggseg* and *ggseg3d*. *Advances in Methods and Practices in Psychological Science*. 2020;3(4):466-483. doi:10.1177/2515245920928009

23. Hinsen K. Dealing with software collapse. *Comput Sci Eng*. 2019;21(3):104-108. doi:10.1109/mcse.2019.2900945

24. Van Vliet M. Seven quick tips for analysis scripts in neuroimaging. *PLoS Comput Biol*. 2020;16(3):e1007358. doi:10.1371/journal.pcbi.1007358

25. Gorgolewski KJ, Poldrack RA. A practical guide for improving transparency and reproducibility in neuroimaging research. *PLoS Biol*. 2016;14(7):e1002506. doi:10.1371/journal.pbio.1002506

26. Quintana DS. A synthetic dataset primer for the biobehavioural sciences to promote reproducibility and hypothesis generation. *eLife*. 2020;9:e53275. doi:10.7554/elife.53275.sa2

27. Chin R, Chang SW, Holmes AJ. Beyond cortex: The evolution of the human brain. *Psychological Review*. Published online 2022.