Reproducible and Programmatic Generation of Neuroimaging Visualizations

Sidhant Chopra *
Department of Psychology
Yale University
CT, USA, 06511
sidhant.chopra@yale.com

Loïc Labache
Department of Psychology
Yale University
CT, USA, 06511

Elvisha Dhamala
Department of Psychology
Yale University
CT, USA, 06511

Edwina R Orchard
Department of Psychology
Yale University
CT, USA, 06511

Avram Holmes
Department of Psychology
Yale University
CT, USA, 06511

August 27, 2022

Neuroimaging visualization forms the centrepiece of quality control, and interpretation and communication of results. Often, these images and figures are produced by manually changing settings on Graphical User Interfaces (GUI). There now exist many well-documented code-based brain visualisation tools that allow users to programmatically generate publication-ready figures directly within environments such as R, Python and MATLAB. Compared to figures generated using GUIs, programmatic tools are more replicable, flexible, interactive, and integrated with the scientific process. This article reviews the advantages of learning and using programmatic neuroimaging visualization tools and provides a list of tools across programming environments. Finally, we introduce examples of specific R and Python tools for visualization of voxel, vertex, region-of-interest, and edge level data.

Keywords Neuroimaging visualization · Reproducibility · Programmatic figures · Open Science · R · Python · Brain Visualization

*Corresponding author

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-based (GUI) 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 exists many code-based software packages which are well-documented and often do not require high-level knowledge of programming, making them more accessible to the neuroimaging community (Table 1). These tools are flexible and allow for the generation of reproducible, high-quality, and publication-ready brain visualizations in only a few lines of code (Figure 1), especially within the R, Python and MATLAB environments. The present article argues for the wide-spread adoption of code-generated visualizations by highlighting major advantages in replicability, flexibility, and integration over GUI based tools. We also providing didactic examples of visualizations and associated code as a reference point.

In brief, by generating visualizations using code, you increase the replicability of your figures for yourself, your collaborators and your readers. Moreover, code-based tools provides more precise controls over display settings, while also benefiting from the advantages of programming, such as the ability to iteratively and rapidly generate multiple figures. Finally, being able to integrate figure generation into your analysis scripts reduces chances of errors and increases the accessibility of resulting analyses pipeline, with more advanced tools now allowing for the seamless integration of prose and code. Here, we review the advantages of learning and using programmatic neuroimaging visualizations, focusing on benefits to replicability, flexibility and integration. We conclude by introducing examples of R-packages which allow for visualization of statistics at region-of-interest (ROI), voxel, vertex, and edge level data, followed by a brief discussion of limitations and functionality gaps in code-based brain visualization tools.

2 Replicability

In recent years, there have been multiple large-scale efforts empirically demonstrating the lack of reproducibility of findings from neuroimaging data (Poldrack et al., 2017). One common solution proposed for achieving robust and reliable discoveries has been to encourage scientific output which 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 increasing transparency and code sharing of neuroimaging analyses, the sharing of code used to generate figures which include 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

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

Visualization scripts 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. By writing and sharing code used to generate brain visualizations, a direct and traceable link is established between the underlying data and the corresponding scientific figure. While the code that produces a replicable figure doesn't necessarily reflect the validity of the scientific finding or the accuracy of the associated content, it allows for reproducibility, instilling transparency and robustness, while demonstrating a desire to further scientific knowledge. Some even consider publishing figures which cannot be replicated as closer to advertising, rather than science (Steel, 2013).

While some GUI-based tools have historically offered command-line access to generate replicable visualizations, they can lack both the flexibility to easily generate publication ready figures and the benefits, such as iteration, provided by your preferred programming environment. Likewise, other GUI-based tools offer replicability in the form of automatically generated batch scripts or in-built terminals, which are often idiosyncratic and lack documentation to make them easily usable or reproducible by those not familiar with the specific software.

3 Flexibility

Being able to exactly replicate your figures via code has marked advantages beyond positive 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 your entire scientific workflow. Changing inputs and settings via code allows for the easy production of multiple figures, such as those resulting from multiple analyses which require similar visualizations. A simple for-loop or copying and pasting the code with altered input 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 re-generated 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. Keep in mind that the gains of writing code for your figures are cumulative, and in addition to improving your programming, you start to build a code-base for figure generation that you can continue to reuse and share throughout your scientific career.

Precise controls via code over visualization settings, such as color schemes, legend placement and camera angles, can provide you with 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 of setting required for creating an image. Most code-based tools also come with documentation, with R-packages on the *CRAN* or *Neuroconductor* (Muschelli et al., 2019) repositories requiring detailed guidance. Recent packages have started to include detailed 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) which provide examples of figures that can be used as starting points or templates for new users. 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 we are not yet at the stage where visualizing the data during processing is no longer necessary. Nonetheless, when working with large datasets such as Human Connectome Project (Van Essen et al., 2013) or UK BioBank (Sudlow et al., 2015), it is unfeasible to use traditional GUI-based tool 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 you to iterate your visualization code over each image of a large datasets making checking the quality of each data processing or analysis step accessible and achievable. The visual outputs of each iteration can be complied into accessible documents which can be easily scrolled, with more advanced usage allowing for the creation of interactive HTML reports, similar to those created by standardized data processing tools like *fmriprep* (Esteban et al., 2019). Increasing capacity to conduct visual quality control on larger datasets can increase the identification of processing errors and result in more reliable and valid findings from your data.

4 Integrative and Interactive Reporting

Often programming languages such as R, Python and MATLAB are used for the analysis and non-brain visualizations in neuroimaging studies, but the brain visualizations resulting from these analyses are outsourced to separate GUI's of tools such as *FSLeyes*, *Freeview* or *ITK-snap*. Switching from your analysis environment to a GUI-based visualization process can be a cumbersome deviation from the scientific workflow. This can make debugging errors more difficult, as you have to regularly switch program to visually examine the results of any modifications or adjustments to prior analyses. Using the brain visualization tools that already exist within your chosen programming environment can provide instant visual feedback on the impact of modifications to processing or analyses.

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 your 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 allowing the submission of reproducible manuscripts, including reproducible figures (e.g. *e-Life*, *Aperture Neuro*), with other journals like *F1000Research* and *GigaScience* even allowing on-demand re-running of code computational environment linked to the associated article using 'compute capsules' from the cloud-based platform Code Ocean (Code Ocean, 2021).

Neuroimaging data are often spatially 3D and can have multiple time points, adding a 4th dimension (e.g., fMRI 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 interpretations. While well-curated 3D renderings can help with spatial localisation (see Madan, 2015; Pernet & Madan, 2019), in the end, static images can only provide an incomplete representation of the data, and forces 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 that you can generate 'rich' media like interactive widgets – figures or animations, which 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 your paper more engaging for the reader. Such rich brain visualizations lend themselves to being embedded or shared on science communication mediums beyond academic papers - such as presentations, websites and social media - all of which can promote the communication of your research with peers and a reach larger audiences (Li and Xie, 2020). This last point is becoming increasingly salient as marketing science on social media has become a core medium for spreading discoveries, science communication to the public, and even a primary avenue for employment opportunities for early-career researchers (Baker, 2015; Lee, 2019). Overall, public engagement is one of the cornerstones of science, and the images we create are at the center of the process.

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	+	+	+		
brainconn				+	
brainR	+		+		

	Voxel	Vertex	ROI	Edge	Streamlines
ciftitools	+	+	+*		
fsbrain	+	+	+*		
ggseg			+		
neurobase	+				
oro.nifti	+				
Python					
ANTsPy	+	+	+		
brainiak	+				
Brainplotlib		+	+*		
Brainspace/surfplot		+	+*		
DIPY	+				+
ENIGMA toolbox			+		
FSLeyes	+	+	+		
ggseg			+		
graphpype				+	
MMVT		+	+	+	
MNE	+	+	+		
mrivis	+				
NaNSlice	+				
netneurotools		+	+*		
nilearn	+	+	+	+	
niwidget	+	+			+
Pycortex	+	+	+*		
pySurfer		+	+*		
surfice	+	+	+	+	+
surfplot		+	+		
Visbrain	+	+	+	+	
MATLAB					
BrainNetViewer	+		+	+	
Brainspace		+	+*		
Brainstorm	+	+	+		

	Voxel	Vertex	ROI	Edge	Streamlines
bspmview	+		+		
CandlabCore	+		+		
ECoG/fMRI Vis toolbox		+	+*		
ENIGMA toolbox			+		
FieldTrip	+	+			
Lead-DBS	+		+		
mni2fs		+			
mrtools	+	+	+*		
plotSurfaceROI-Boundary		+	+*		
Vistasoft	+	+	+		+

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. This list does not include cross-platform general purpose visualization software.
Cortex only

5 Examples of Neuroimaging Visualization Packages available in *R and Python*

The following four sections provide brief examples of well documented and beginner friendly packages and functions available in R and Python for visualizing voxel, vertex, ROI and edge-level data. This is not an exhaustive list of packages available for visualizing brain data in R and Python (See Table 1), rather, the following sections aim to give the reader a sense of the available options, and an entry point to using them. All code used to compile the figures, as well as the contents of each panel are provided in the Supplement or can be accessed in the accompanying Git repository.

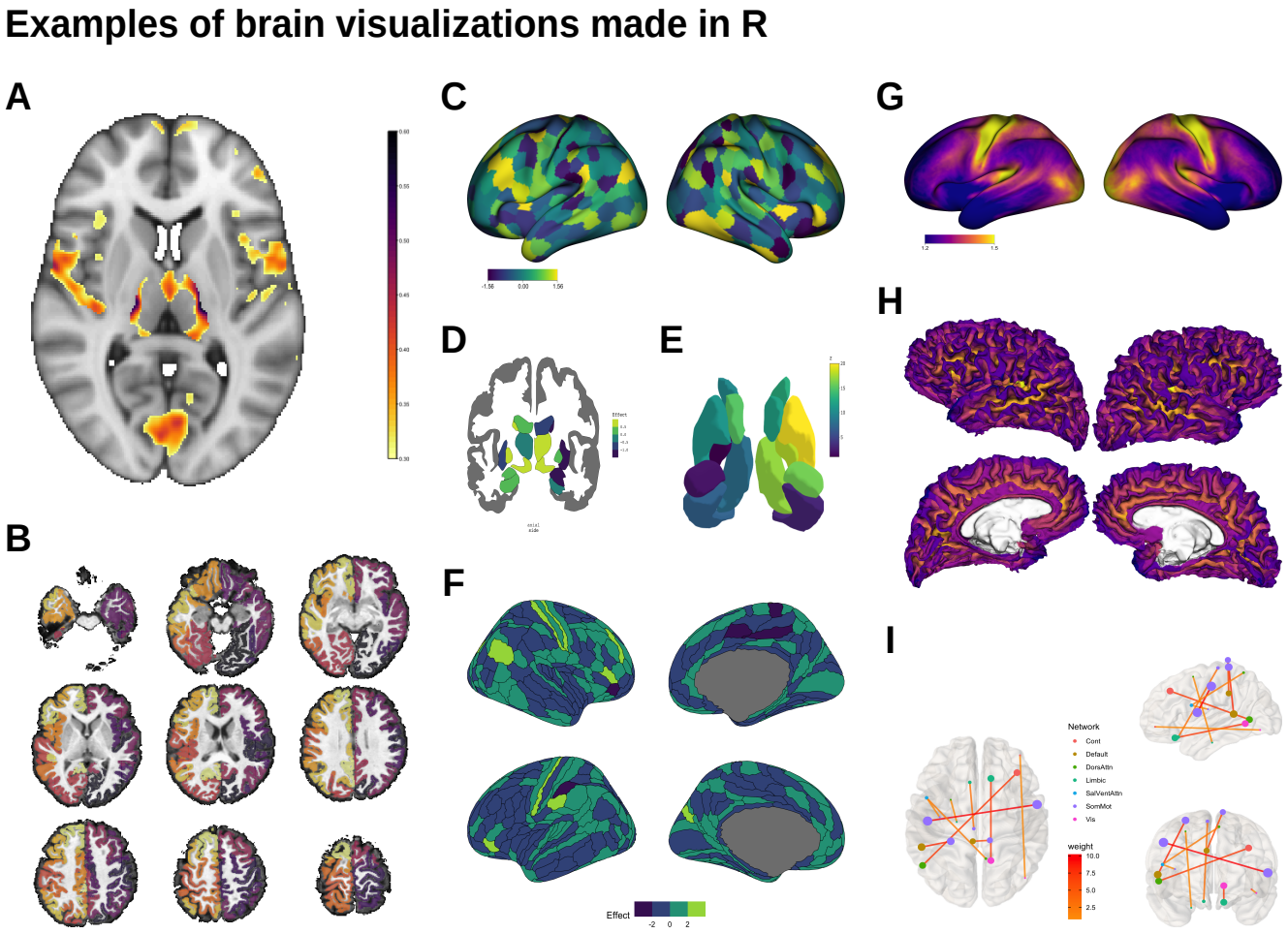

Figure 1. Examples of brain imaging visualization make in using different R packages. A) Voxel-based statistical map threshold and overlaid over a T1-weighted template data, with a single axial slice shown. Made using the the `ortho2` function from the `neurobase` package. B) A voxel-level cortical parcellation overlaid on T1-weighted MRI data 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 regions, 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 statistical 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 statistical 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 template cortical surface where a statistical 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 template of a brain rendering. 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 the Supplement.

Figure 1. Examples of brain imaging visualization make in using different Python packages. A)

5.1 Voxel-Level Visualizations.

R. The `oro.nifti` and `neurobase` packages allow for the reading, writing, manipulation and visualization of voxel-level imaging data of either *nifti*, *analyze* or *afni* formats. In particular, the `image`, `slice` and `ortho2` functions allow users to display the desired number of slices in the desired orientations, while also providing precise control over overlaid images, color scales, legend placement and other aesthetic settings (Fig 1A-B). Smoothing and re-sampling of images is sometimes required for visualization purposes, which can be achieved via the `ANTsR` or `fslr` packages.

Python. The `NiBable` library provides the ability to read in and covert many common neuroimaging file formats, including *nifti,* to data-structures which can be manipulated and visualized. While these data-structures often form 3D or 4D matrices, so can be visualized using commonly used general purpose visualization tools such as `matplotlib` , purpose built librires like `nilearn` provide functions like `plot_stat_map` and `plot_glass_brain` which are user friendly and can generate publication ready figures (Fig. X). `nilearn` also includes smoothing and re-sampling functions which can sometimes required for visualization purposes.

5.2 Vertex-Level Visualizations.

R. The recently developed `fsbrain` package allows for the visualization of vertex-level and cortical ROI-level data which were derived using *FreeSurfer*. The package contains many flexible functions for visualizing individual-level and group-level measures such as cortical thickness, volume or surface area (Fig 1F). In addition to producing publication-ready figures, the visualization functions can allow for visual quality control of large datasets processed using FreeSurfer. The package also comes with extensive guides and documentations to assist users (Schäfer & Ecker, 2020).

CIFTI or 'grey-ordinate' data is becoming a popular format for structural and functional imaging data, as it combines vertex-level data of the cortex with voxel-level data for the cerebellum and sub-cortex. In R, CIFTI data can now be read, visualized and manipulated using the well-documented `ciftiTools` package (Fig 1C,1G; Pham, Muschelli, & Mejia, 2022). This package allows for both vertex-level visualization of the surface as well as voxel-level visualization of cerebellum and subcortex.

Python. ciftify.

5.3 ROI-Level Visualizations.

R. Statistical parameters can be mapped onto discreet cortical and sub-cortical regions using the `ggseg` and `ggseg3d` packages (Mowinckel & Vidal-Piñeiro, 2020). These packages flexibly generates aesthetic renderings of cortical, sub-cortical and white matter ROIs in 2D (Fig 1D,1F) and 3D (Fig 1E). Usually, ROI-level interpretations are dependent on a standardized atlas or parcellation scheme of the brain and accordingly, this package and its sister-packages (`ggsegExtra`) provide a large array of commonly used atlases

and some functionality for users to contribute other atlases. Additionally, being part of the 'Grammar of Graphics' framework (Wickham, 2016) allows for integration with packages such as `gganimate`, enabling users to animate and visualize dynamic changes in spatial and temporal statistics across ROIs. Notably, the `ciftiTools` package also provides functions to visualize surface and voxel-level ROIs for the cortex and sub-cortex respectively (Fig 1E), flexibly allowing CIFTI format atlases and ROIs to be displayed.

Python.

5.4 Edge-Level Visualizations.

Analyses which divide the brain into discrete regions and examine pair-wise dependencies between regional phenotypes are becoming increasingly common. Often the results of these dependency analyses are statistics relating to 'edges' or links between any two regions. The package `brainconn` allows users to plot brain regions as nodes of a network graph, and dependencies between regions as edges of that graph, following a standardized coordinate space in both 2D and interactive 3D (Figure 1I). The graphs can be weighted, with the strength of the edges represented with color or line thickness, as well as directed, where arrows will be used in place of edges. This package comes with node coordinates corresponding to multiple commonly used atlases and contains functionality for users to easily enter their own custom atlas coordinates.

Python.

6 Limitations and Functionality Gaps

Most of the tools introduced above do not require a strong knowledge of programming, but there is still a steeper learning curve when compared to use a GUI. This is especially true when learning how to make fine adjustments to visual auxiliary such as legend placement, font size and multi-panel figure positioning, for the purpose of a publication-ready figure. While most code-based tools offer some control over these finer steps, there are differences between them in feature availability and usability. Relatedly, while some interactive image viewers which can be opened within an integrated development environment (e.g. Muschelli, 2016), for quick and interactive viewing of single images, GUI tool can be faster and more practical.

Often cerebellar and brain-stem regions are not well represented in software (e.g. Figure 1), potentially mirroring the cortico-centric sentiment that has prevailed in human neuroimaging research (Chin, Chang, & Holmes, 2022). Likewise, custom non-cortical atlases such as non-standard subcortical atlas schemes are not yet straight forward, and usually require multiple functions and packages to visualize. Finally, some neuroimaging related data types, such as streamlines resulting from DWI-based tractography, are still not well represented in code-based visualization tools.

As can be seen in Table 1, there are usually multiple packages within each programming environment which are capable of visualizing each data type. While this provide choice for advanced users, it can also lead to

confusion for novice users who may not be familiar with the nuanced differences between tools. Future work should continue to consolidate brain visualization methods into unified beginner-friendly code-based tools which are capable of plotting multiple data types.

7 References

Poldrack, R. A., Baker, C. I., Durnez, J., Gorgolewski, K. J., Matthews, P. M., Munafò, M. R., ... & Yarkoni, T. (2017). Scanning the horizon: towards transparent and reproducible neuroimaging research. Nature reviews neuroscience, 18(2), 115-126.

Steel, G. (2013, September 3). Publishing research without data is simply advertising, not science. Open Knowledge Foundation. `https://blog.okfn.org/2013/09/03/publishing-research-without-data-is-simply-advertising-not-science/`

The Comprehensive R Archive Network. (n.d.). Retrieved April 21, 2022, from `https://cran.r-project.org/`

Muschelli, J., Gherman, A., Fortin, J. P., Avants, B., Whitcher, B., Clayden, J. D., ... & Crainiceanu, C. M. (2019). Neuroconductor: an R platform for medical imaging analysis. *Biostatistics*, *20*(2), 218-239.

Pham, D., Muschelli, J., & Mejia, A. (2022). ciftiTools: A package for reading, writing, visualizing, and manipulating CIFTI files in R. *NeuroImage*, 118877.

Mowinckel, A. M., & Vidal-Piñeiro, D. (2020). Visualization of brain statistics with R packages ggseg and ggseg3d. *Advances in Methods and Practices in Psychological Science*, *3*(4), 466-483.

Huntenburg, J., Abraham, A., Loula, J., Liem, F., Dadi, K., & Varoquaux, G. (2017). Loading and plotting of cortical surface representations in Nilearn. *Research Ideas and Outcomes*, *3*, e12342.

Wickham H. (2016). *ggplot2: Elegant Graphics for Data Analysis.* Springer-Verlag New York. ISBN 978-3-319-24277-4, `https://ggplot2.tidyverse.org`.

Muschelli, J. (2016). *papayar*. GitHub repository. `https://github.com/muschellij2/papayar`

David C. Van Essen, Stephen M. Smith, Deanna M. Barch, Timothy E.J. Behrens, Essa Yacoub, Kamil Ugurbil, for the WU-Minn HCP Consortium. (2013). The WU-Minn Human Connectome Project: An overview. NeuroImage 80(2013):62-79.

Sudlow, C., Gallacher, J., Allen, N., Beral, V., Burton, P., Danesh, J., ... & Collins, R. (2015). UK biobank: an open access resource for identifying the causes of a wide range of complex diseases of middle and old age. PLoS medicine, 12(3), e1001779.

Esteban, O., Markiewicz, C. J., Blair, R. W., Moodie, C. A., Isik, A. I., Erramuzpe, A., ... & Gorgolewski, K. J. (2019). fMRIPrep: a robust preprocessing pipeline for functional MRI. Nature methods, 16(1), 111-116.

Pernet, C., & Madan, C. R. (2019). Data visualization for inference in tomographic brain imaging.

Code Ocean. (2021, October 27). Computational Research Platform on. Retrieved April 21, 2022, from `https://codeocean.com/`

Madan, C. R. (2015). Creating 3D visualizations of MRI data: A brief guide. F1000Research, 4.

Baker M. (2015). Social media: A network boost. Nature 518: 263–265. 10.1038/nj7538-263a

Lee J.-S. (2019). How to use Twitter to further your research career. Nature. 10.1038/d41586-019-00535-w

Li, Y., & Xie, Y. (2020). Is a picture worth a thousand words? An empirical study of image content and social media engagement. Journal of Marketing Research, 57(1), 1-19.

Schäfer, T., & Ecker, C. (2020). fsbrain: an R package for the visualization of structural neuroimaging data. bioRxiv.

Chin, R., Chang, S. W., & Holmes, A. J. (2022). Beyond cortex: The evolution of the human brain. *Psychological Review*.

8 Supplement (Code used to generate Figures)

```
#Figure 1A (Voxel)

library(neurobase)

#Load in a nifti (.nii.gz) file of a standardized template so use as a background.
template <- readnii('data/MNI152_T1_1mm_brain.nii.gz') #load the nifti file into R

#Load in a nifti (.nii.gz) file of the statistic you wanted of overlay, with the same dimensions as the
effect <- readnii("data/MNI152_effect_size.nii.gz")

#Threshold the statistic map so only values about a specified amount are displayed
effect[effect<0.3] <- NA

#Set the breaks/intervals you want on the color bar (e.g. from .3 to .6, by intervals of 0.005)
breaks = seq(.3,.6, by=0.005)

#[Optional] If you want to output the figure as a .png, open a png image device
png("data/fig1a.png")

#Use the ortho2 function from the neurobase package to make the figure
ortho2(x = template,   #Specify the background template
       y = effect,     #Specify the effect you want to overlay on the template
       crosshairs = F, #Remove the cross-hairs
       bg = "white",   #Make the background white
       NA.x = T,       #Do not display NA values
       col.y= rev(viridis::inferno(n = 500)), #Select the color scale. In this case I have used the inf
       xyz = c(70,50,80), #set the x y & z slice you want to visualize
       useRaster = T,     #Sometimes using Raster makes for clearer plots
       ycolorbar = TRUE,  #Add a color bar
       mfrow = c(1,1)) +  #layout of brain views (for this figure, we just want a axial view)
  colorbar(breaks, col = rev(viridis::inferno(n=length(breaks)-1)), #add a color bar with same color sc
           text.col = "black", labels = TRUE, maxleft = 0.95)  #Set the specifications for the color ba

dev.off()
```

```
#Figure 1B (Voxel)

library(neurobase)
library(scales)

#Load in a nifti (.nii.gz) file to use as a background.
t1 <- readnii('data/sub-001__t1_warped.nii.gz')

#Load in a nifti (.nii.gz) file to use as a overlay.
atlas <-  readnii('data/sub-001_schaefer300n7_aseg_to_dwispace_gm_rois.nii.gz')

png('data/fig1b.png')
overlay(x=robust_window(t1), y=atlas,
        plot.type = "single",
        z=c(seq(30,110,10)),
        col.y =  alpha(viridis::inferno(300), 0.6),
        plane = "axial",
        useRaster = T,
        bg = "white",
        NA.x=T,
```

```
        zlim.y =c(1,300))
dev.off()
```

```
#vertex 1
#Figure 1C
library(rgl)
library(ggplot2)
# Load the package and point to the Connectome Workbench
library(ciftiTools)

#Need connectome workbench to use ciftiTools - set path below
ciftiTools.setOption("wb_path", "/Applications/workbench/")

# Read in CIFTI file
cifti_fname <- ciftiTools::ciftiTools.files()$cifti["dtseries"]
surfL_fname <- ciftiTools.files()$surf["left"]
surfR_fname <- ciftiTools.files()$surf["right"]

cii <- read_cifti(
  cifti_fname, brainstructures="all",
  surfL_fname=surfL_fname,
  surfR_fname=surfR_fname)

#Plot cifti surface
view_xifti_surface(cii,
                   hemisphere = "both",
                   view = "lateral",
                   idx = 1,
                   colors = viridis::plasma(n = 100),
                   zlim = c(1.2,1.5),
                   legend_embed = T,
                   cex.title = 2)



rgl.snapshot("data/fig1c.png")
rgl.close()
```

```
#Vertex 2
#Figure 1X - Remove white space


library(fsbrain)
library(rgl)

#download_optional_data()
#download_fsaverage(accept_freesurfer_license = TRUE)

subjects_dir <- get_optional_data_filepath("subjects_dir")

subject_id <- 'subject1'

colourmap <- colorRampPalette(viridis::plasma(n = 1000))
fsbrain.set.default.figsize(700, 700);

brain = vis.subject.morph.native(subjects_dir,
                                 subject_id, 'sulc',
                                 cortex_only = T,
                                 views=NULL,
```

```
                                draw_colorbar = FALSE,
                                makecmap_options = list('colFn'=colourmap))

img = export(brain, draw_colorbar = FALSE, output_img = "data/fig1d.png")
```

```
#ROI1
#Figure 1e
library(ciftiTools)
parc <- load_parc("Yeo_7")
view_xifti_surface(parc,
                   legend_embed = T,
                   hemisphere = "right")

rgl.snapshot("data/fig1e.png")

rgl.close()
```

```
#ROI1
#Figure 1X - combine with previous figure

set.seed(1993) #set a random seed (good practice for reproducibility)

parc <- load_parc("Schaefer_400")

ramdom_metric <- rnorm(400)

cii <- move_from_mwall(cii, NA)

parc_vec <- c(as.matrix(parc))

xii_metric <- c(NA, ramdom_metric)[parc_vec + 1]

xii1 <- select_xifti(cii, 1)

xii_metric <- newdata_xifti(xii1, xii_metric)

plot2 <- view_xifti_surface(xii_metric,
                            colors = viridis::viridis(n = 400),
                            borders = "black",
                            hemisphere = "both",
                            view = 'lateral')

rgl.snapshot("data/fig1f.png")

rgl.close()
```

```
#ROI2

#devtools::install_github("LCBC-UiO/ggsegGlasser")
library(ggseg)
library(ggplot2)
library(ggsegGlasser)

set.seed(1993) #set a random seed (good practice for reproducibility)


base_atlas <- as.data.frame(na.omit(cbind(glasser$data$region, glasser$data$hemi)))

colnames(base_atlas) <- c("region", "hemi")
```

```
Effect <- rnorm(dim(base_atlas)[1]) #generate a random numbers for each roi in the atlas

base_atlas <- cbind(Effect, base_atlas)


cortex <- ggseg(atlas = glasser,
                .data = base_atlas,
                mapping=aes(fill=Effect),
                position="stacked",
                colour="black",
                size=.2,
                show.legend = T,
                plot.background = "white") +
  scale_fill_viridis_b() +
   theme_void() +
    theme(legend.position = "bottom")

sub_base_atlas <- as.data.frame(na.omit(cbind(aseg$data$region, aseg$data$hemi)))

colnames(sub_base_atlas) <- c("region", "hemi")

Effect <- rnorm(dim(sub_base_atlas)[1])

sub_base_atlas <- cbind(Effect, sub_base_atlas)

subcortex <- ggseg(atlas = aseg,
                   .data = sub_base_atlas,
                   mapping=aes(fill=Effect),
                   position = "dispersed",
                   colour="black",
                   hemi =c('left', 'right'),
                   size=.2,
                   show.legend = T,
                   plot.background = "white") +
  scale_fill_viridis_b()

ggsave(filename = 'data/fig1g1.png', plot = cortex, device = 'png', bg = 'white')
ggsave(filename = 'data/fig1g2.png', plot = subcortex, device = 'png', bg = 'white')
```

```
make_ggseg3d <- function(attribute,
                         colour.pal = c("light yellow",
                                        "orange",
                                        "red",
                                        "dark red"),
                         hide.colourbar=FALSE,
                         output.png=FALSE,
                         file.name="ggseg_3d.png") {
  # Inputs:
  #' attribute = This must be a numeric vector in this: Left-Thalamus, Left-Caudate, Left-Putamen, Left

  library(ggseg3d)
  library(tidyr)
  library(dplyr)
  #remove(aseg_3d)
  aseg_3d <- aseg_3d
  aseg_3d <- tidyr::unnest(aseg_3d, cols = c(ggseg_3d))

  attribute.ggseg3d <- c(rep(NA, 4), attribute[1:4], rep(NA, 3), attribute[5:7],
```

```
                          rep(NA, 5), attribute[8:14],  rep(NA, 6))

  data <- dplyr::mutate(aseg_3d, attribute =  attribute.ggseg3d)
  #remove NA regions
  aseg_3d[which(is.na(data$attribute)),] <- NA
  aseg_3d <- tidyr::drop_na(aseg_3d)

  data[which(is.na(data$attribute)),] <- NA
  data <- tidyr::drop_na(data)
  data$attribute[data$attribute==0]<-NA #make 0 valus NA so they are set as grey in ggseg3d

  scene=list(camera = list(eye = list(x = 0, y = 1, z = -2.25)),
             aspectratio = list(x=1.6,y=1.6,z=1.6))


  plot <- ggseg3d::ggseg3d(.data = data,
                           atlas = aseg_3d,
                           colour = "attribute",
                           text = "attribute",
                           palette = colour.pal)
  plot <- remove_axes(plot)
  plot <- plotly::layout(plot,
                         scene = scene,
                         width = 600, height = 600)

  if(hide.colourbar==TRUE){
    plot <- plotly::hide_colorbar(plot)
  }

  if( output.png==TRUE){
    plotly::orca(plot, file =  file.name)
  }

  return(plot)

}

set.seed(1993)
volume <- sample(1:20, 14) #randomly generated data for 14 structures
make_ggseg3d(volume,
             colour.pal = viridis::viridis(n=length(volume)),
             output.png = T, file.name = "data/fig1g3.png")
```

```
#Edge (also add 3d version)
#Fig1X

#devtools::install_github("sidchop/brainconn")
library(brainconn)

conmat <- example_weighted_undirected
degree <- rowSums(conmat)[which(rowSums(conmat)!=0)]

brainconn(atlas ="schaefer300_n7",
                   conmat=conmat,
                   node.size = degree/2,
                   view="top",
                   edge.color.weighted = T,
                   background.alpha = 0.8,
```

```
                  show.legend = F,
                  edge.color = scale_edge_colour_viridis())



ggsave(filename = 'data/fig1h.png',
       plot = fig1h,
       device = 'png',
       bg = 'white',
       height = 7,
       units = "in")


brainconn3D(atlas ="schaefer300_n7",
            conmat=conmat,
            show.legend = F)
```