Team A Code

Amina Asghar
Nasri Binsaleh
Onintsoa Ramananandroniaina

Reference code: https://carpentries-incubator.github.io/SDC-BIDS-fMRI/aio/index.html

In [1]:
```
# laod required packages
import sys                              # for installing package
import os                               # for os related functions
import numpy as np                      # for dealing with numpy array
import pandas as pd                     # for dealing with dataframe
import matplotlib.pyplot as plt         # for plotting
%matplotlib inline
import nibabel as nib                   # for dealing with fMRI data
from numpy.linalg import inv            # for inv()
%matplotlib inline
from scipy import stats                 # for stats function e.g. to find p-value
import nilearn
from nilearn import plotting            # for plotting brain map
import math                             # for math function, checking NaN
import sklearn as sk
import warnings
warnings.filterwarnings('ignore')
```

Inspecting data

Warning

Not all subjects/sessions/runs have the same scanning parameters.

LOCATION:/sub-control08/func/sub-control08_task-nonmusic_run-5_bold.nii.gz

REASON: The most common set of dimensions is: 80,80,50,105 (voxels), This file has the dimensions: 80,80,50,71 (voxels).

LOCATION: /sub-mdd05/func/sub-mdd05_task-nonmusic_run-1_bold.nii.gz

REASON: The most common set of dimensions is: 80,80,50,105 (voxels), This file has the dimensions: 80,80,50,89 (voxels).

In [2]:
```
# importing the functional MRI data file
fmri_file = 'Dataset/dataset/sub-control01/func/sub-control01_task-music_run-1_bold.nii.
```

In [3]:
```
# check the functional MRI data shape, voxel size, and units
f_img = nib.load(fmri_file)
print(f_img.shape)
print(f_img.header.get_zooms())
print(f_img.header.get_xyzt_units())
```

```
(80, 80, 50, 105)
(2.9, 2.9, 3.0, 3.0)
('mm', 'sec')
```

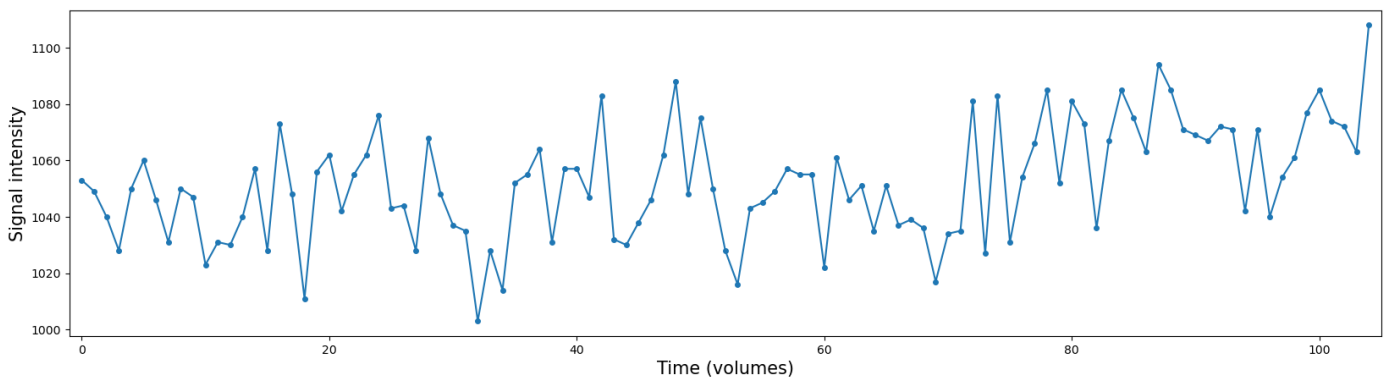
In [4]:
```
# getting fMRI data
f_img_data = f_img.get_fdata()
print(f_img_data.shape)
```

```
(80, 80, 50, 105)
```

In [5]:
```
# extract the time series of the middle voxel
mid_vox_ts = f_img_data[39, 39, 24, :]
print("Voxel timeseries shape: %s" % (mid_vox_ts.shape,))
```

```
Voxel timeseries shape: (105,)
```

In [6]:
```
# time-by-time fluctuations in signal intensity by
# plotting the time series directly
plt.figure(figsize=(20, 5))
plt.plot(mid_vox_ts, 'o-', ms=4)
plt.xlim(-1, mid_vox_ts.size)
#plt.ylim(38000, 55000)
plt.ylabel('Signal intensity', fontsize=15)
plt.xlabel('Time (volumes)', fontsize=15)
plt.show()
```



Constructing GLM

In [7]:
```
# get signal for a voxel
voxel_signal = f_img_data[39, 39, 24, :]
# get the events
events_df = pd.read_csv('Dataset/dataset/sub-control01/func/sub-control01_task-music_run
                        sep='\t')
events_df
```

Out[7]:

	onset	duration	trial_type
0	0.0	33.0	tones
1	33.0	3.0	response
2	36.0	31.5	negative_music
3	67.5	3.0	response
4	70.5	31.5	tones
5	102.0	3.0	response
6	105.0	31.5	positive_music
7	136.5	3.0	response
8	139.5	31.5	tones
9	171.0	3.0	response
10	174.0	31.5	negative_music
11	205.5	3.0	response
12	208.5	31.5	tones

13	240.0	3.0	response
14	243.0	31.5	positive_music
15	274.5	3.0	response
16	277.5	31.5	tones
17	309.0	3.0	response

In [8]:
```
## record the response onset in arrays
response_onsets = events_df[events_df['trial_type'] ==
                            'negative_music']
response_onsets = response_onsets.append(events_df[events_df['trial_type'] ==
                                                   'positive_music'])
onsets = response_onsets['onset'].to_numpy()
onsets = onsets.astype(int)
onsets
```

Out[8]:
```
array([ 36, 174, 105, 243])
```

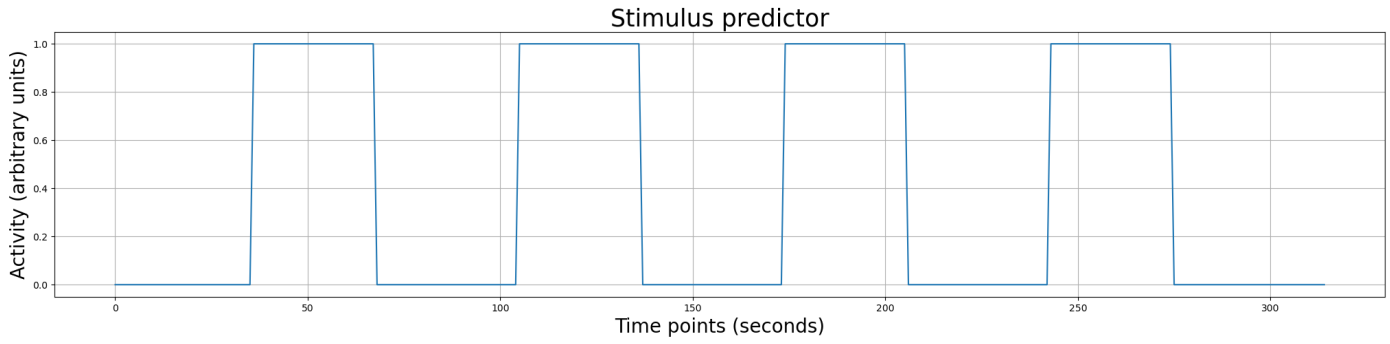
In [9]:
```
## make a range of onsets
onset_range = []
for onset in onsets:
    onset_range = onset_range + list(range(onset,onset+32))
```

In [10]:
```
## convert the onset array to a proper predictor
## (with the same shape as the fMRI signal)
# length of experiment is 105 volume with TR = 3s
predictor = np.zeros(105*3)
predictor[onset_range] = 1  # set the predictor at the indices to 1
print("Shape of predictor: %s" % (predictor.shape,))
print("\nContents of predictor array:\n%r" % predictor.T)
```

```
Shape of predictor: (315,)

Contents of predictor array:
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [11]:
```
## visualizing the predictor
plt.figure(figsize=(25, 5))
#plt.plot(predictor_congruent, marker='o')
plt.plot(predictor, c='tab:blue')
plt.xlabel('Time points (seconds)', fontsize=20)
plt.ylabel('Activity (arbitrary units)', fontsize=20)
#plt.xlim(0, 300)
```

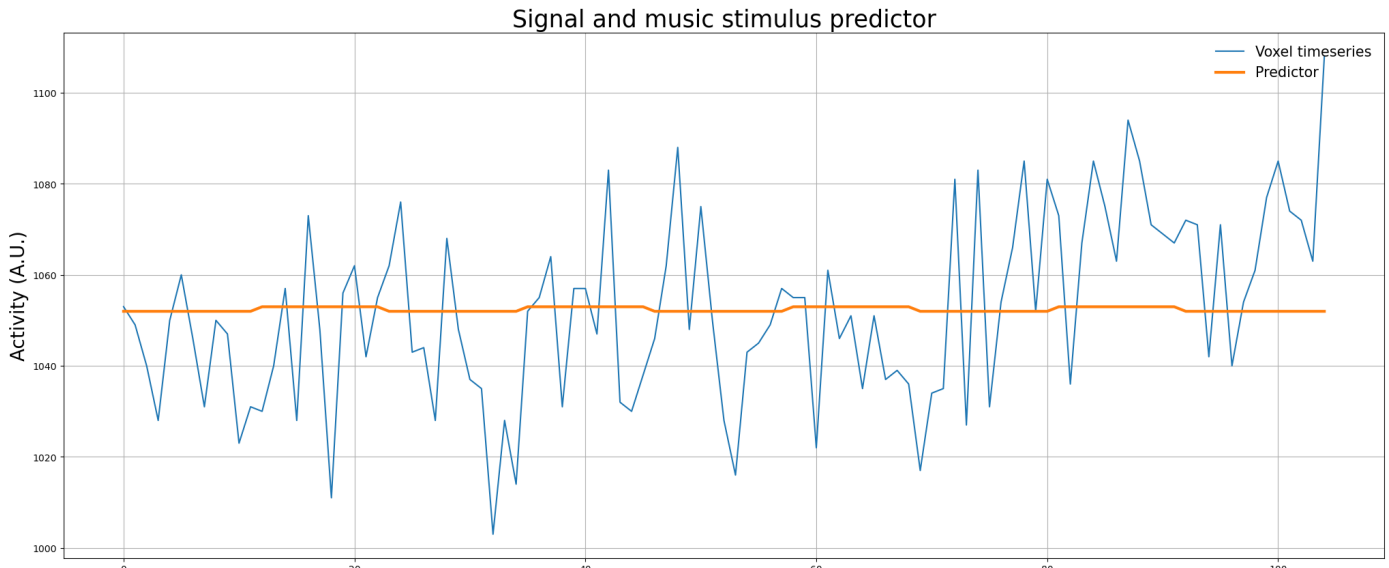
```
#plt.ylim(-.5, 1.5)
plt.title('Stimulus predictor', fontsize=25)
plt.grid()
plt.show()
```



In [12]:
```
## since the voxel in fMRI was measured in volume with TR of 3 seconds,
## we have to down sampling the predictor to fit the same unit of time.
from scipy.interpolate import interp1d

original_scale = np.arange(0, 105*3, 1) # from 0 to 105*3 seconds
print("Original scale has %i datapoints (0-315, in seconds)" %
      original_scale.size)
resampler = interp1d(original_scale, predictor)
desired_scale = np.arange(0, 105*3, 3)
print("Desired scale has %i datapoints (0, 3, 6, ... 105, in volumes)" %
      desired_scale.size)
predictor_ds = resampler(desired_scale)
print("Downsampled predictor has %i datapoints (in volumes)" %
      predictor_ds.size)
print(predictor_ds)
```

```
Original scale has 315 datapoints (0-315, in seconds)
Desired scale has 105 datapoints (0, 3, 6, ... 105, in volumes)
Downsampled predictor has 105 datapoints (in volumes)
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

In [13]:
```
## inspecting the predictor and the actual signal before
## moving on to linear regression
plt.figure(figsize=(25, 10))
plt.plot(voxel_signal)
plt.plot(predictor_ds + voxel_signal.mean(), lw=3)
plt.xlabel('Time (in volumes)', fontsize=20)
plt.ylabel('Activity (A.U.)', fontsize=20)
plt.legend(['Voxel timeseries', 'Predictor'], fontsize=15,
           loc='upper right', frameon=False)
plt.title("Signal and music stimulus predictor", fontsize=25)
plt.grid()
plt.show()
```

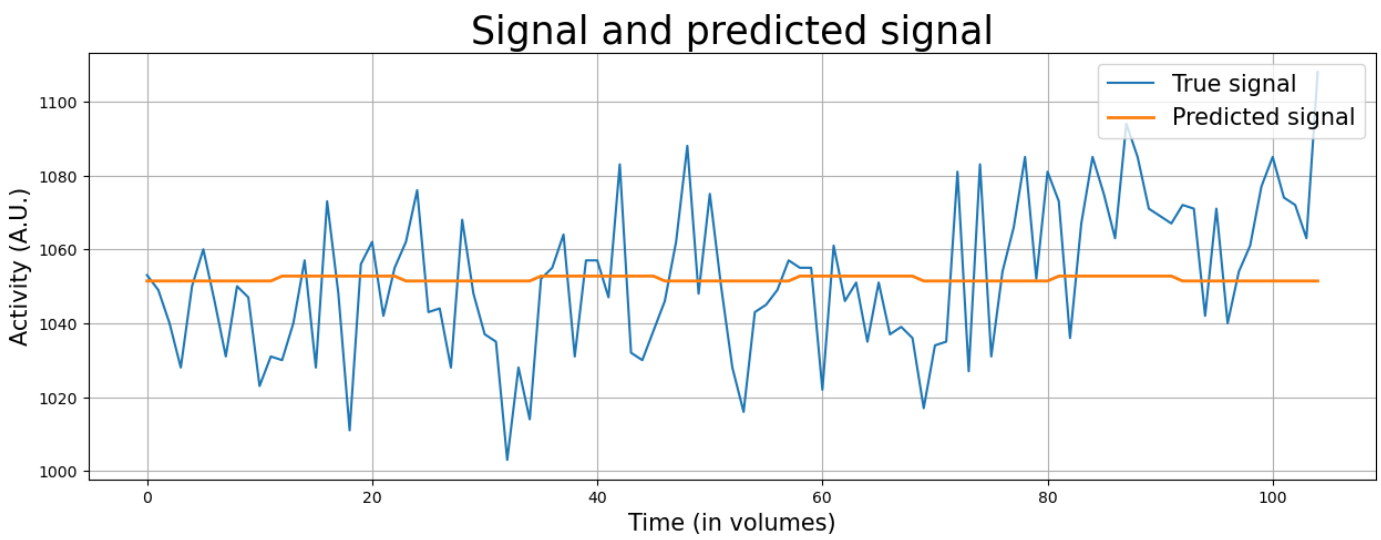


In [14]:
```
# Regression
if predictor_ds.ndim == 1:  # This adds a singleton dimension,
                            # such that you can call np.hstack on it
    predictor_ds = predictor_ds[:, np.newaxis]

icept = np.ones((predictor_ds.size, 1))
X_simple = np.hstack((icept, predictor_ds))
betas_simple = inv(X_simple.T @ X_simple) @ X_simple.T @ voxel_signal
y_hat = X_simple[:, 0] * betas_simple[0] + X_simple[:, 1] * betas_simple[1]
print(betas_simple)
numerator = np.sum((voxel_signal - y_hat) ** 2)
denominator = np.sum((voxel_signal - np.mean(voxel_signal)) ** 2)
r_squared = 1 - numerator / denominator
print('The R² value is: %.3f' % r_squared)
```

```
[1051.42622951    1.32377049]
The R² value is: 0.001
```

In [15]:
```
# function for plotting predicted signal vs. actual signal
def plot_signal_and_predicted_signal(y, X, x_lim=None, y_lim=None):
    """ Plots a signal and its GLM prediction. """
    des = np.hstack((np.ones((y.size, 1)), X))
    betas_simple = np.linalg.lstsq(des, y, rcond=None)[0]
    plt.figure(figsize=(15, 5))
    plt.plot(y)
    plt.plot(des @ betas_simple, lw=2)
    plt.xlabel('Time (in volumes)', fontsize=15)
    plt.ylabel('Activity (A.U.)', fontsize=15)

#     if x_lim is not None:
#         plt.xlim(x_lim)

#     if y_lim is not None:
#         plt.ylim(y_lim)

    plt.legend(['True signal', 'Predicted signal'],
               loc='upper right',
               fontsize=15)
    plt.title("Signal and predicted signal", fontsize=25)
    plt.grid()
    plt.show()
```

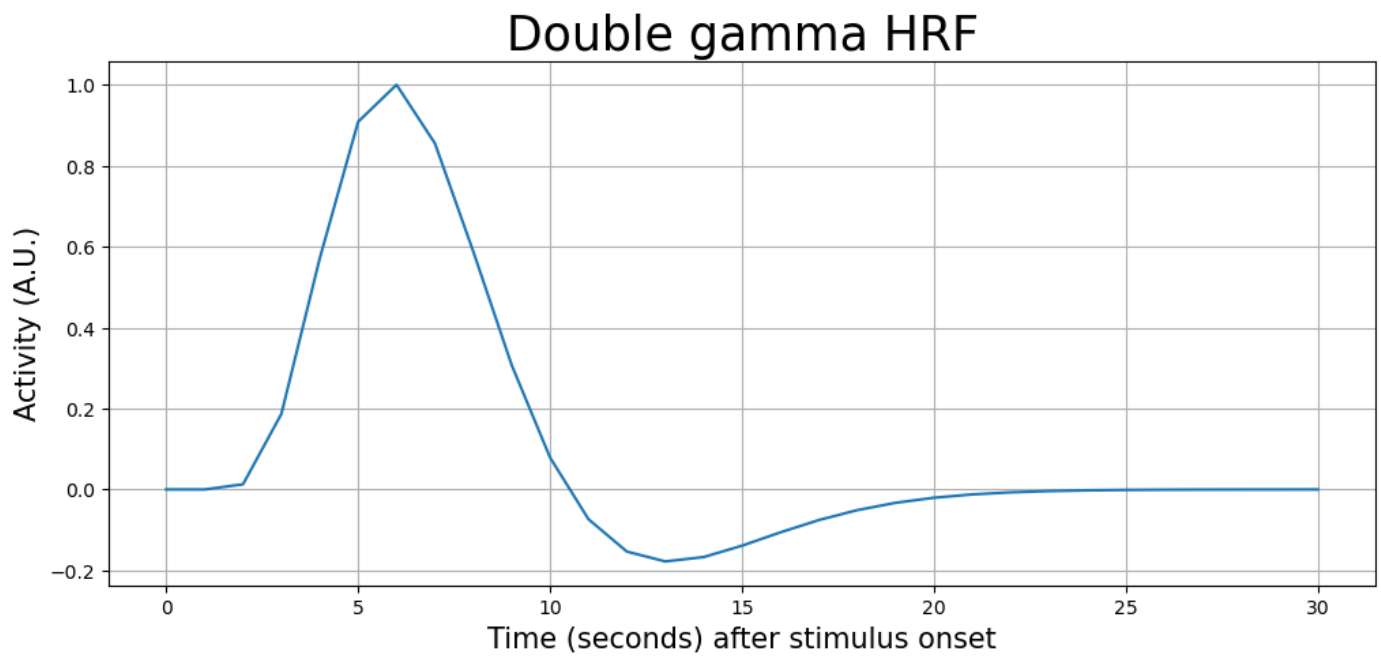
In [16]:
```
# plotting predicted signal vs. actual signal
plot_signal_and_predicted_signal(voxel_signal, predictor_ds)
```



In [17]:
```
# Now, model with BOLD response function
# install nilearn package
# import sys
# !{sys.executable} -m pip install nilearn

# use glover_hrf from nilearn
from nilearn.glm.first_level.hemodynamic_models import glover_hrf

# creating the canonical HFR
TR = 3
osf = 3
length_hrf = 31 # sec # have to shape it according to our data
                # (let's say to match the ITI of the experiment)
canonical_hrf = glover_hrf(tr=TR, oversampling=osf,
                           time_length=length_hrf,
                           onset=0)
canonical_hrf /= canonical_hrf.max()
print("Size of canonical hrf variable: %i" % canonical_hrf.size)
```

```
Size of canonical hrf variable: 31
```

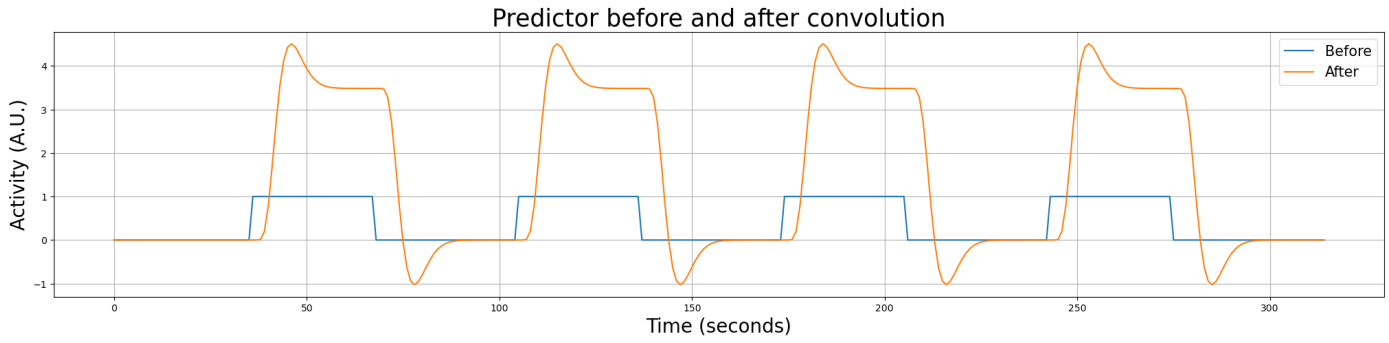
In [18]:
```
# visualizing the canonical HRF
t = np.arange(0, canonical_hrf.size)
plt.figure(figsize=(12, 5))
plt.plot(t,canonical_hrf)
plt.xlabel('Time (seconds) after stimulus onset', fontsize=15)
plt.ylabel('Activity (A.U.)', fontsize=15)
plt.title('Double gamma HRF', fontsize=25)
plt.grid()
plt.show()
```



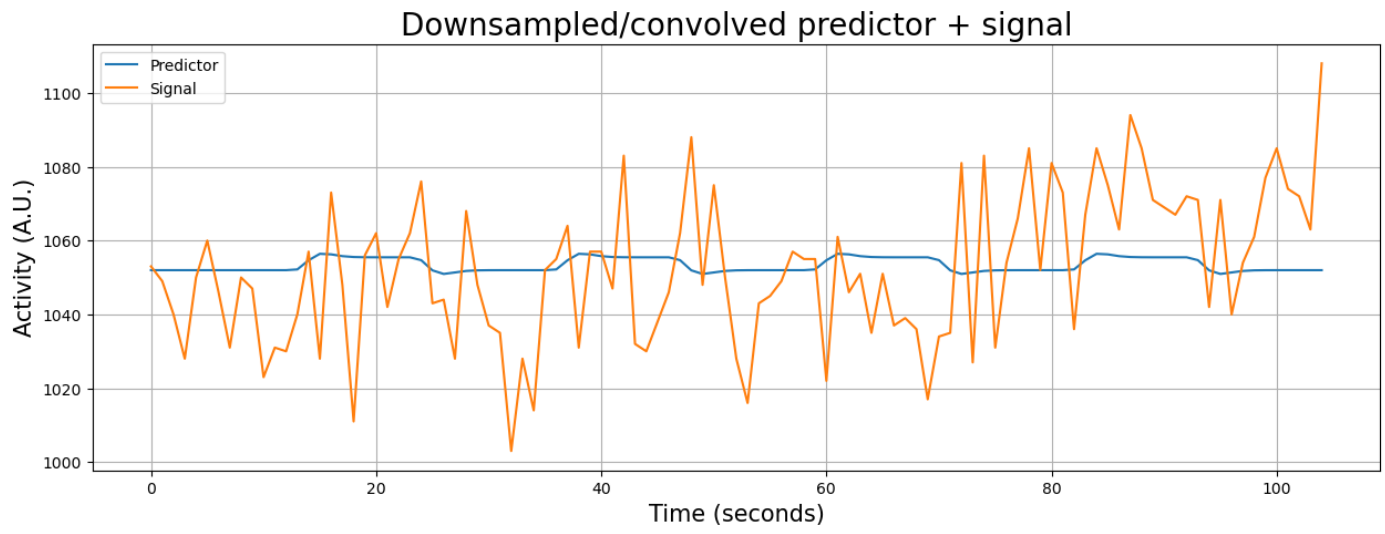
In [19]:
```
# convolve the predictor with the HRF
predictor_conv = np.convolve(predictor.squeeze(), canonical_hrf)
print("The shape of the convolved predictor after convolution: %s" %
      (predictor_conv.shape,))

# After convolution, we also neem to "trim" off
# some excess values from the convolved signal
predictor_conv = predictor_conv[:predictor.size]
print("After trimming, the shape is: %s" %
      (predictor_conv.shape,))

# And we have to add a new axis again to go from shape (N,) to (N, 1),
# which is important for stacking the intercept later
predictor_conv = predictor_conv[:, np.newaxis]
print("Shape after adding the new axis: %s" %
      (predictor_conv.shape,))
```

```
The shape of the convolved predictor after convolution: (345,)
After trimming, the shape is: (315,)
Shape after adding the new axis: (315, 1)
```

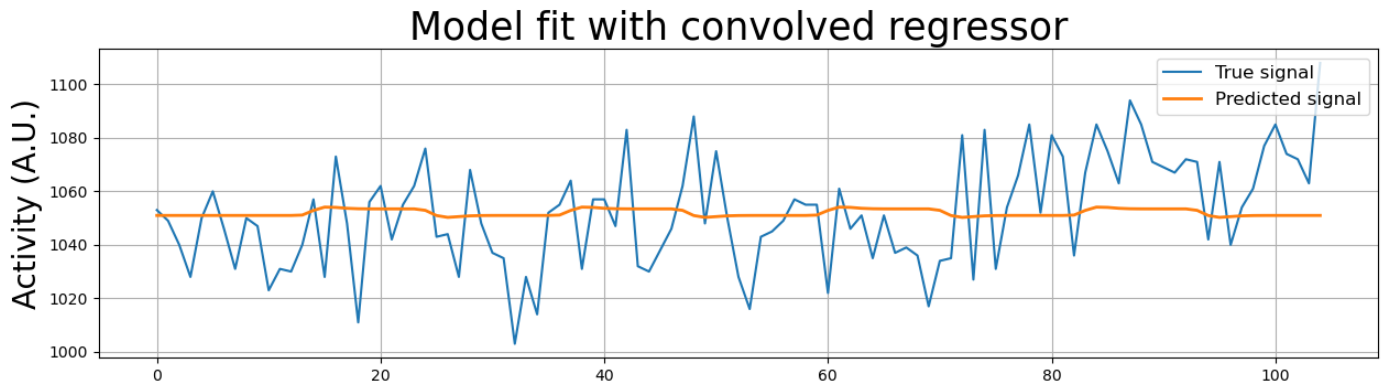
In [20]:
```
# visualizing the predictor before and after the convolution
# congruent
plt.figure(figsize=(25, 5))
plt.plot(predictor)
plt.plot(predictor_conv)
#plt.xlim(-1, 800)
plt.title("Predictor before and after convolution", fontsize=25)
plt.xlabel("Time (seconds)", fontsize=20)
plt.ylabel("Activity (A.U.)", fontsize=20)
plt.legend(['Before', 'After'], loc='upper right', fontsize=15)
plt.grid()
plt.show()
```



In [21]:
```
# scale the convolved predictor back to the scale of volume
original_scale = np.arange(0, 105*3, 1) # from 0 to 105*3 seconds
resampler = interp1d(original_scale, np.squeeze(predictor_conv))
desired_scale = np.arange(0, 105*3, 3)
predictor_conv_ds = resampler(desired_scale)
# x_lim, y_lim = (0, 400), (990, 1020)
plt.figure(figsize=(15, 5))
plt.plot(predictor_conv_ds + voxel_signal.mean())
plt.plot(voxel_signal)
plt.grid()
plt.title('Downsampled/convolved predictor + signal', fontsize=20)
plt.ylabel('Activity (A.U.)', fontsize=15)
plt.xlabel('Time (seconds)', fontsize=15)
plt.legend(['Predictor', 'Signal'])
# plt.xlim(x_lim)
plt.show()
```



In [22]:
```
# Now, fitting convolved predictor in GLM
if predictor_conv_ds.ndim == 1:
    # Add back a singleton axis (which was removed before downsampling)
    # otherwise stacking will give an error
    predictor_conv_ds = predictor_conv_ds[:, np.newaxis]

intercept = np.ones((predictor_conv_ds.size, 1))
X_conv = np.hstack((intercept, predictor_conv_ds))
betas_conv = inv(X_conv.T @ X_conv) @ X_conv.T @ voxel_signal
plt.figure(figsize=(15, 8))
plt.subplot(2, 1, 1)
plt.plot(voxel_signal)
plt.plot(X_conv @ betas_conv, lw=2)
# plt.xlim(x_lim)
plt.ylabel("Activity (A.U.)", fontsize=20)
plt.title("Model fit with convolved regressor", fontsize=25)
plt.legend(['True signal', 'Predicted signal'], fontsize=12,
           loc='upper right')
plt.grid()
```



In [23]:
```
# Evaluate the model
from numpy.linalg import lstsq
# numpy implementation of OLS, because we're lazy
y_hat_conv = X_conv @ betas_conv
y_hat_orig = X_simple @ lstsq(X_simple, voxel_signal, rcond=None)[0]
MSE_conv = ((y_hat_conv - voxel_signal) ** 2).mean()
MSE_orig = ((y_hat_orig - voxel_signal) ** 2).mean()
print("MSE of model with convolution is %.3f while the MSE of the model without convolut
 (MSE_conv, MSE_orig))
R2_conv = 1 - (np.sum((voxel_signal - y_hat_conv) ** 2) /
               np.sum((voxel_signal - voxel_signal.mean()) ** 2))
R2_orig = 1 - (np.sum((voxel_signal - y_hat_orig) ** 2) /
               np.sum((voxel_signal - voxel_signal.mean()) ** 2))
print("R-squared of model with convolution is %.5f and without convolution it is %.5f."
 (R2_conv, R2_orig))
```

```
MSE of model with convolution is 398.470 while the MSE of the model without convolution
is 399.725.
R-squared of model with convolution is 0.00420 and without convolution it is 0.00107.
```

T-maps Music

Sub-control01 Run-1

In [24]:
```
# importing the functional MRI data file
fmri_file = 'Dataset/dataset/sub-control01/func/sub-control01_task-music_run-1_bold.nii.
f_img = nib.load(fmri_file)
# getting fMRI data
f_img_data = f_img.get_fdata()
```

In [25]:
```
def design_variance(X, which_predictor=1): # X : Numpy Array of shape (N, P)
    is_single = isinstance(which_predictor, int)
    if is_single:
        idx = which_predictor
    else:
        idx = np.array(which_predictor) != 0

    c = np.zeros(X.shape[1])
    c[idx] = 1 if is_single == 1 else which_predictor[idx]
    des_var = c.dot(np.linalg.inv(X.T.dot(X))).dot(c.T)
    return des_var
```

In [26]:
```
# get a statistical map (a for loop to compute t-value of every voxel)
t_map = f_img_data[:, :, :, 0]  # to store t-values of each voxel (3D map)
for i in range(f_img_data.shape[0]):
    for j in range(f_img_data.shape[1]):
        for k in range(f_img_data.shape[2]):
```

```
            voxel_signal = f_img_data[i, j, k, :]
            # start the regression
            if predictor_conv_ds.ndim == 1:
                # Add back a singleton axis
                #(which was removed before downsampling)
                # otherwise stacking will give an error
                predictor_conv_ds = predictor_conv_ds[:, np.newaxis]
            # linear regression
            intercept = np.ones((predictor_conv_ds.size, 1))
            X_conv = np.hstack((intercept, predictor_conv_ds))
            betas_conv = inv(X_conv.T @ X_conv) @ X_conv.T @ voxel_signal
            # get design variance
            design_variance_predictor = design_variance(X_conv,
                                                        which_predictor=1)
            # get degree of freedom and noise terms
            y_hat = X_conv @ betas_conv
            N = voxel_signal.size
            P = X_conv.shape[1]
            df = (N - P)  # degree of freedom
            sigma_hat = np.sum((voxel_signal - y_hat) ** 2) / df # noise
            #get t-value
            t = betas_conv[1] / np.sqrt(sigma_hat *
                                        design_variance_predictor)
            if math.isnan(t):
                t = 0
            # store the t-value in t-map
            t_map[i, j, k] = round(t, 3)
```

In [27]:
```
t_map
```

Out[27]:
```
array([[[ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ],
        [ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ],
        [ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ],
        ...,
        [ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ],
        [ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ],
        [ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ]],

       [[ 0.613, -0.644,  1.372, ...,  0.004,  0.363,  0.117],
        [ 1.311,  0.339,  0.391, ..., -0.286,  0.629,  0.258],
        [ 0.425, -1.058,  1.886, ..., -0.873, -0.139,  1.298],
        ...,
        [-1.761, -1.145,  0.513, ..., -0.708,  0.4  ,  1.096],
        [-0.751, -0.579,  0.814, ...,  1.793,  2.232, -0.936],
        [ 1.391, -1.322, -0.046, ...,  0.075, -0.561,  0.142]],

       [[-0.61 , -0.555,  2.   , ...,  1.486, -0.067,  0.132],
        [ 0.666,  0.158,  0.731, ...,  0.565, -0.675,  0.477],
        [-0.593, -1.193,  2.069, ..., -1.729, -1.478, -2.209],
        ...,
        [-0.212, -0.162, -0.259, ...,  1.131,  1.307,  1.685],
        [ 0.191,  1.291, -0.084, ...,  1.563,  0.557,  1.281],
        [-1.686, -1.097,  0.216, ...,  1.116, -0.565,  0.859]],

       ...,

       [[ 0.792,  1.338,  1.315, ...,  1.517,  1.428, -0.388],
        [ 0.235,  0.258,  0.542, ...,  0.385,  0.517, -1.358],
        [ 0.243,  1.396,  1.052, ..., -0.31 ,  0.728, -1.789],
        ...,
        [-0.35 , -1.627, -0.783, ..., -1.083, -1.711, -0.608],
        [-0.091,  0.394, -0.009, ..., -0.24 , -0.995, -0.008],
        [-0.368, -0.018,  0.191, ...,  1.364, -0.248,  0.089]],

       [[-0.159, -1.021,  0.843, ..., -0.771,  0.105, -2.62 ],
```

```
 [ 0.06 ,  0.996, -0.383, ..., -0.477,  0.185,  0.022],
 [ 0.21 , -1.22 ,  0.618, ...,  0.132,  1.704, -0.217],
 ...,
 [ 0.121,  1.584, -1.263, ..., -0.918, -0.651,  1.561],
 [ 1.37 ,  0.214, -0.79 , ..., -1.988,  0.725,  0.59 ],
 [-0.568,  1.813,  0.362, ..., -1.037, -1.087, -0.07 ]],

[[ 0.038, -2.643,  1.441, ..., -0.73 ,  0.13 ,  1.221],
 [ 1.722, -1.98 ,  0.092, ..., -1.223, -0.386,  1.247],
 [-0.691, -1.76 ,  1.326, ..., -0.418,  2.266, -3.137],
 ...,
 [ 0.421, -0.016, -0.783, ..., -0.461, -0.894, -1.002],
 [-1.852,  0.536, -0.408, ...,  1.385,  1.503, -0.663],
 [-0.169,  0.386,  0.791, ...,  0.334,  1.335,  1.708]]])
```

In [28]:
```
## convert t-map to NifTi image

# load the data
func = nib.load(fmri_file)

# to save this 3D (ndarry) numpy use this
ni_img = nib.Nifti1Image(t_map, func.affine)
```

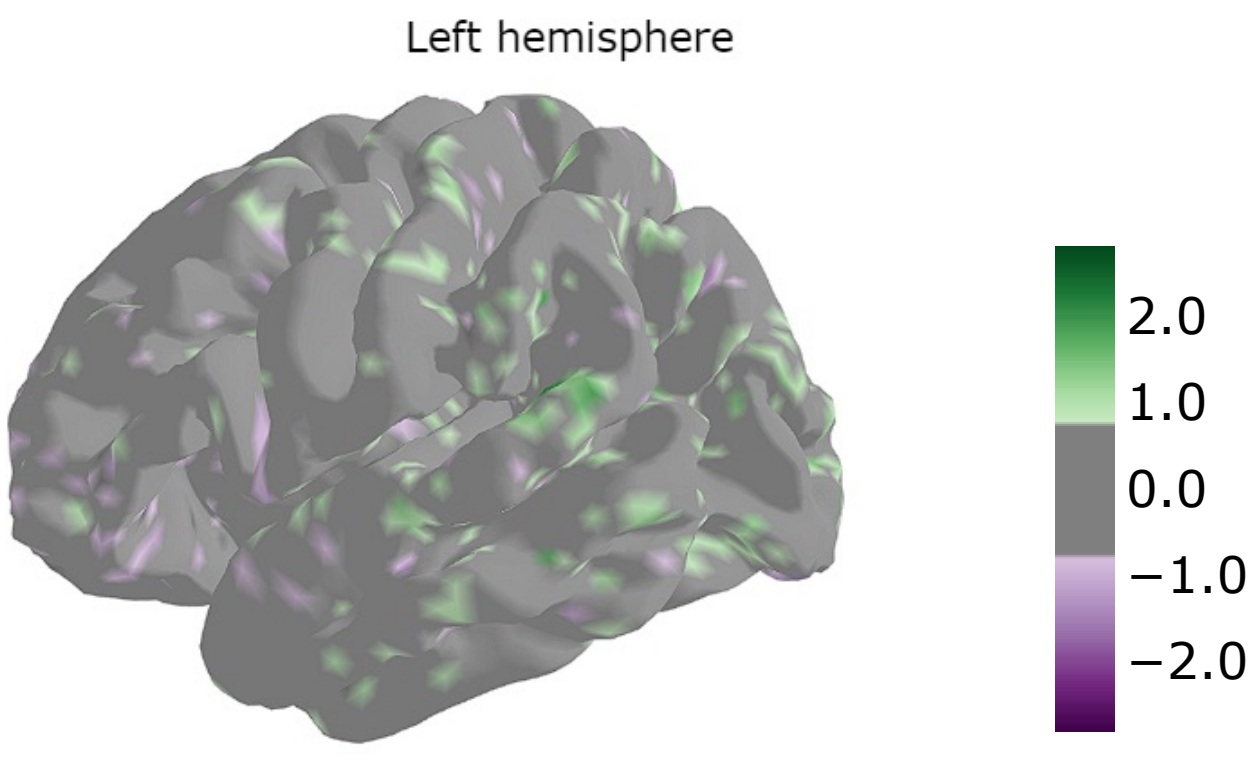
In [29]:
```
# Get a cortical mesh
from nilearn import datasets
fsaverage = datasets.fetch_surf_fsaverage()
```

In [30]:
```
print(fsaverage['description'])
```

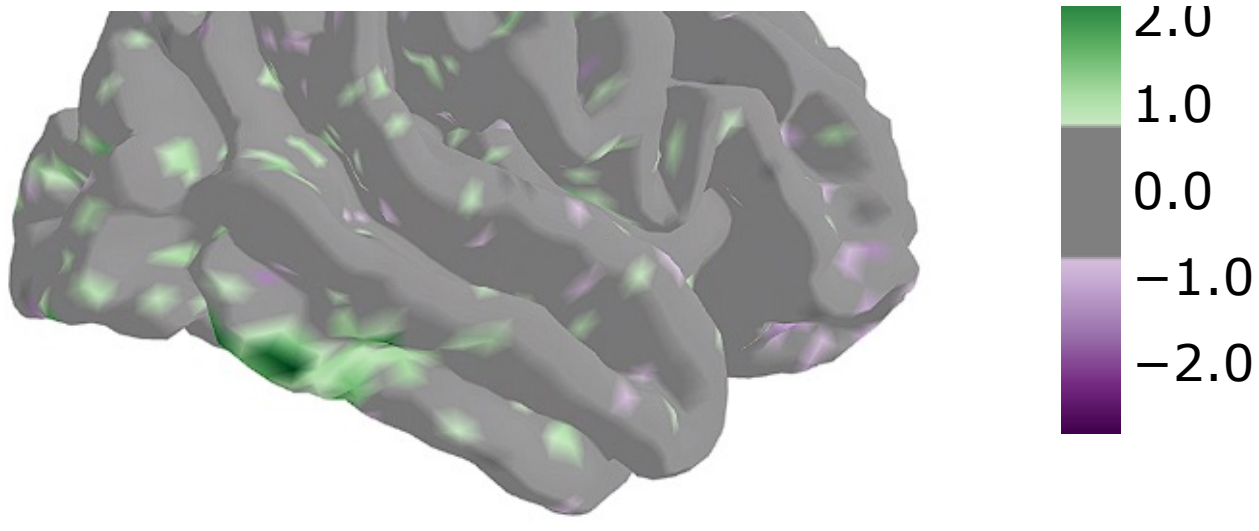
```
fsaverage5


Notes
-----
Fsaverage5 standard surface as distributed with Freesurfer (Fischl et al, 1999)

Content
-------
    :'area_left': Gifti file, left hemisphere area data
    :'area_right': Gifti file, right hemisphere area data
    :'curv_left': Gifti file, left hemisphere curvature data
    :'curv_right': Gifti file, right hemisphere curvature data
    :'pial_left': Gifti file, left hemisphere pial surface mesh
    :'pial_right': Gifti file, right hemisphere pial surface mesh
    :'infl_left': Gifti file, left hemisphere inflated pial surface mesh
    :'infl_right': Gifti file, right hemisphere inflated pial
                   surface mesh
    :'sphere_left': Gifti file, left hemisphere sphere surface mesh
    :'sphere_right': Gifti file, right hemisphere sphere surface mesh
    :'sulc_left': Gifti file, left hemisphere sulcal depth data
    :'sulc_right': Gifti file, right hemisphere sulcal depth data
    :'thick_left': Gifti file, left hemisphere cortical thickness data
    :'thick_right': Gifti file, right hemisphere cortical thickness data
    :'white_left': Gifti file, left hemisphere white surface mesh
    :'white_right': Gifti file, right hemisphere white surface mesh

References
----------
Fischl et al, (1999). High-resolution intersubject averaging and a
coordinate system for the cortical surface. Hum Brain Mapp 8, 272-284.
```

In [31]:
```
# Sample the 3D data around each node of the mesh
from nilearn import surface
texture = surface.vol_to_surf(ni_img, fsaverage.pial_right)
```

In [32]:
```
# plot the result
from nilearn import plotting

# fig = plotting.plot_surf_stat_map(
#     fsaverage.infl_right, texture, hemi='right',
#     title='Surface right hemisphere', colorbar=True,
#     threshold=0,  engine='plotly', bg_map=fsaverage.sulc_right,
#     engine='plotly'
# )

fig = plotting.plot_surf_stat_map(fsaverage.pial_left, texture,
                                  cmap = 'PRGn', engine='plotly',
                                  threshold="90%", title='Left hemisphere')
fig.show()
```



In [33]:
```
fig = plotting.plot_surf_stat_map(fsaverage.pial_right, texture,
                                  cmap = 'PRGn', engine='plotly',
                                  threshold="90%", title='Right hemisphere')
fig.show()
```



Sub-control02 Run-3

In [34]:
```
# importing the functional MRI data file
fmri_file = 'Dataset/dataset/sub-control02/func/sub-control02_task-music_run-3_bold.nii.

# getting fMRI data
f_img_data = f_img.get_fdata()
```

In [35]:
```
def design_variance(X, which_predictor=1): # X : Numpy Array of shape (N, P)
    is_single = isinstance(which_predictor, int)
    if is_single:
        idx = which_predictor
    else:
        idx = np.array(which_predictor) != 0

    c = np.zeros(X.shape[1])
    c[idx] = 1 if is_single == 1 else which_predictor[idx]
    des_var = c.dot(np.linalg.inv(X.T.dot(X))).dot(c.T)
    return des_var
```

In [36]:
```
# get a statistical map (a for loop to compute t-value of every voxel)
t_map = f_img_data[:, :, :, 0]  # to store t-values of each voxel (3D map)
for i in range(f_img_data.shape[0]):
    for j in range(f_img_data.shape[1]):
        for k in range(f_img_data.shape[2]):
            voxel_signal = f_img_data[i, j, k, :]
            # start the regression
            if predictor_conv_ds.ndim == 1:
                # Add back a singleton axis (which was removed before downsampling)
                # otherwise stacking will give an error
                predictor_conv_ds = predictor_conv_ds[:, np.newaxis]
            # linear regression
            intercept = np.ones((predictor_conv_ds.size, 1))
            X_conv = np.hstack((intercept, predictor_conv_ds))
            betas_conv = inv(X_conv.T @ X_conv) @ X_conv.T @ voxel_signal
            # get design variance
            design_variance_predictor = design_variance(X_conv,
                                                        which_predictor=1)
            # get degree of freedom and noise terms
```

```
            y_hat = X_conv @ betas_conv
            N = voxel_signal.size
            P = X_conv.shape[1]
            df = (N - P)  # degree of freedom
            sigma_hat = np.sum((voxel_signal - y_hat) ** 2) / df # noise
            #get t-value
            t = betas_conv[1] / np.sqrt(sigma_hat * design_variance_predictor)
            if math.isnan(t):
                t = 0
            # store the t-value in t-map
            t_map[i, j, k] = round(t, 3)
```

In [37]:
```
## convert t-map to NifTi image

# load the data
func = nib.load(fmri_file)

# to save this 3D (ndarry) numpy use this
ni_img = nib.Nifti1Image(t_map, func.affine)
```

In [38]:
```
# Get a cortical mesh
from nilearn import datasets
fsaverage = datasets.fetch_surf_fsaverage()
```

In [39]:
```
# Sample the 3D data around each node of the mesh
from nilearn import surface
texture = surface.vol_to_surf(ni_img, fsaverage.pial_right)
```

plot the result from nilearn import plotting fig = plotting.plot_surf_stat_map(fsaverage.pial_left, texture, cmap = 'PRGn', engine='plotly') fig.show()fig = plotting.plot_surf_stat_map(fsaverage.pial_right, texture, cmap = 'PRGn', engine='plotly', threshold=0) fig.show()

Sub-mdd01 Run-1

In [40]:
```
# importing the functional MRI data file
fmri_file = 'Dataset/dataset/sub-mdd01/func/sub-mdd01_task-music_run-1_bold.nii.gz'
f_img = nib.load(fmri_file)
# getting fMRI data
f_img_data = f_img.get_fdata()
```

In [41]:
```
def design_variance(X, which_predictor=1): # X : Numpy Array of shape (N, P)
    is_single = isinstance(which_predictor, int)
    if is_single:
        idx = which_predictor
    else:
        idx = np.array(which_predictor) != 0
    c = np.zeros(X.shape[1])
    c[idx] = 1 if is_single == 1 else which_predictor[idx]
    des_var = c.dot(np.linalg.inv(X.T.dot(X))).dot(c.T)
    return des_var
```

In [42]:
```
# get a statistical map (a for loop to compute t-value of every voxel)
t_map = f_img_data[:, :, :, 0]  # to store t-values of each voxel (3D map)
for i in range(f_img_data.shape[0]):
    for j in range(f_img_data.shape[1]):
        for k in range(f_img_data.shape[2]):
            voxel_signal = f_img_data[i, j, k, :]
            # start the regression
            if predictor_conv_ds.ndim == 1:
                # Add back a singleton axis (which was removed before downsampling)
                # otherwise stacking will give an error
```

```
            predictor_conv_ds = predictor_conv_ds[:, np.newaxis]
        # linear regression
        intercept = np.ones((predictor_conv_ds.size, 1))
        X_conv = np.hstack((intercept, predictor_conv_ds))
        betas_conv = inv(X_conv.T @ X_conv) @ X_conv.T @ voxel_signal
        design_variance_predictor = design_variance(X_conv, which_predictor=1)  # ge
        # get degree of freedom and noise terms
        y_hat = X_conv @ betas_conv
        N = voxel_signal.size
        P = X_conv.shape[1]
        df = (N - P)  # degree of freedom
        sigma_hat = np.sum((voxel_signal - y_hat) ** 2) / df # noise
        #get t-value
        t = betas_conv[1] / np.sqrt(sigma_hat * design_variance_predictor)
        if math.isnan(t):
            t = 0
            # store the t-value in t-map
            t_map[i, j, k] = round(t, 3)
```

In [43]:
```
## convert t-map to NifTi image

# load the data
func = nib.load(fmri_file)

# to save this 3D (ndarry) numpy use this
ni_img = nib.Nifti1Image(t_map, func.affine)
```

In [44]:
```
# Get a cortical mesh
from nilearn import datasets
fsaverage = datasets.fetch_surf_fsaverage()
```

In [45]:
```
# Sample the 3D data around each node of the mesh
from nilearn import surface
texture = surface.vol_to_surf(ni_img, fsaverage.pial_right)
```

plot the result from nilearn import plotting fig = plotting.plot_surf_stat_map(fsaverage.pial_left, texture, cmap = 'PRGn', engine='plotly', threshold="90%", title='Left hemisphere') fig.show()fig = plotting.plot_surf_stat_map(fsaverage.pial_right, texture, cmap = 'PRGn', engine='plotly', threshold="90%", title='Right hemisphere') fig.show()

Sub-mdd03 Run-1

In [46]:
```
# importing the functional MRI data file
fmri_file = 'Dataset/dataset/sub-mdd03/func/sub-mdd03_task-music_run-1_bold.nii.gz'

# getting fMRI data
f_img_data = f_img.get_fdata()
```

In [47]:
```
def design_variance(X, which_predictor=1): # X : Numpy Array of shape (N, P)
    is_single = isinstance(which_predictor, int)
    if is_single:
        idx = which_predictor
    else:
        idx = np.array(which_predictor) != 0

    c = np.zeros(X.shape[1])
    c[idx] = 1 if is_single == 1 else which_predictor[idx]
    des_var = c.dot(np.linalg.inv(X.T.dot(X))).dot(c.T)
    return des_var
```

In [48]:
```
# get a statistical map (a for loop to compute t-value of every voxel)
t_map = f_img_data[:, :, :, 0]  # to store t-values of each voxel (3D map)
for i in range(f_img_data.shape[0]):
```

```
    for j in range(f_img_data.shape[1]):
        for k in range(f_img_data.shape[2]):
            voxel_signal = f_img_data[i, j, k, :]
            # start the regression
            if predictor_conv_ds.ndim == 1:
                # Add back a singleton axis (which was removed before downsampling)
                # otherwise stacking will give an error
                predictor_conv_ds = predictor_conv_ds[:, np.newaxis]
            # linear regression
            intercept = np.ones((predictor_conv_ds.size, 1))
            X_conv = np.hstack((intercept, predictor_conv_ds))
            betas_conv = inv(X_conv.T @ X_conv) @ X_conv.T @ voxel_signal
            design_variance_predictor = design_variance(X_conv, which_predictor=1)  # ge
            # get degree of freedom and noise terms
            y_hat = X_conv @ betas_conv
            N = voxel_signal.size
            P = X_conv.shape[1]
            df = (N - P)  # degree of freedom
            sigma_hat = np.sum((voxel_signal - y_hat) ** 2) / df # noise
            #get t-value
            t = betas_conv[1] / np.sqrt(sigma_hat * design_variance_predictor)
            if math.isnan(t):
                t = 0
            # store the t-value in t-map
            t_map[i, j, k] = round(t, 3)
```

In [49]:
```
## convert t-map to NifTi image

# load the data
func = nib.load(fmri_file)

# to save this 3D (ndarry) numpy use this
ni_img = nib.Nifti1Image(t_map, func.affine)
```

In [50]:
```
# Get a cortical mesh
from nilearn import datasets
fsaverage = datasets.fetch_surf_fsaverage()
```

In [51]:
```
# Sample the 3D data around each node of the mesh
from nilearn import surface
texture = surface.vol_to_surf(ni_img, fsaverage.pial_right)
```

plot the result from nilearn import plotting fig = plotting.plot_surf_stat_map(fsaverage.pial_left, texture, cmap = 'PRGn', engine='plotly') fig.show()fig = plotting.plot_surf_stat_map(fsaverage.pial_right, texture, cmap = 'PRGn', engine='plotly', threshold="95%", title='Right hemisphere') fig.show()

Sub-control16 Run-1

In [52]:
```
# importing the functional MRI data file
fmri_file = 'Dataset/dataset/sub-control16/func/sub-control16_task-music_run-3_bold.nii.

# getting fMRI data
f_img_data = f_img.get_fdata()
```

In [53]:
```
def design_variance(X, which_predictor=1): # X : Numpy Array of shape (N, P)
    is_single = isinstance(which_predictor, int)
    if is_single:
        idx = which_predictor
    else:
        idx = np.array(which_predictor) != 0

    c = np.zeros(X.shape[1])
```

```
    c[idx] = 1 if is_single == 1 else which_predictor[idx]
    des_var = c.dot(np.linalg.inv(X.T.dot(X))).dot(c.T)
    return des_var
```

In [54]:
```
# get a statistical map (a for loop to compute t-value of every voxel)
t_map = f_img_data[:, :, :, 0]  # to store t-values of each voxel (3D map)
for i in range(f_img_data.shape[0]):
    for j in range(f_img_data.shape[1]):
        for k in range(f_img_data.shape[2]):
            voxel_signal = f_img_data[i, j, k, :]
            # start the regression
            if predictor_conv_ds.ndim == 1:
                # Add back a singleton axis (which was removed before downsampling)
                # otherwise stacking will give an error
                predictor_conv_ds = predictor_conv_ds[:, np.newaxis]
            # linear regression
            intercept = np.ones((predictor_conv_ds.size, 1))
            X_conv = np.hstack((intercept, predictor_conv_ds))
            betas_conv = inv(X_conv.T @ X_conv) @ X_conv.T @ voxel_signal
            design_variance_predictor = design_variance(X_conv, which_predictor=1)  # ge
            # get degree of freedom and noise terms
            y_hat = X_conv @ betas_conv
            N = voxel_signal.size
            P = X_conv.shape[1]
            df = (N - P)  # degree of freedom
            sigma_hat = np.sum((voxel_signal - y_hat) ** 2) / df # noise
            #get t-value
            t = betas_conv[1] / np.sqrt(sigma_hat * design_variance_predictor)
            if math.isnan(t):
                t = 0
            # store the t-value in t-map
            t_map[i, j, k] = round(t, 3)
```

In [55]:
```
## convert t-map to NifTi image

# load the data
func = nib.load(fmri_file)

# to save this 3D (ndarry) numpy use this
ni_img = nib.Nifti1Image(t_map, func.affine)
```

In [56]:
```
# Get a cortical mesh
from nilearn import datasets
fsaverage = datasets.fetch_surf_fsaverage()
```

In [57]:
```
# Sample the 3D data around each node of the mesh
from nilearn import surface
texture = surface.vol_to_surf(ni_img, fsaverage.pial_right)
```

plot the result from nilearn import plotting fig = plotting.plot_surf_stat_map(fsaverage.pial_left, texture, cmap = 'PRGn', engine='plotly') fig.show()fig = plotting.plot_surf_stat_map(fsaverage.pial_right, texture, cmap = 'PRGn', engine='plotly', threshold=0) fig.show()

Sub-mdd14 Run-1

In [58]:
```
# importing the functional MRI data file
fmri_file = 'Dataset/dataset/sub-mdd14/func/sub-mdd14_task-music_run-3_bold.nii.gz'

# getting fMRI data
f_img_data = f_img.get_fdata()
```

In [59]:
```
def design_variance(X, which_predictor=1): # X : Numpy Array of shape (N, P)
```

```
    is_single = isinstance(which_predictor, int)
    if is_single:
        idx = which_predictor
    else:
        idx = np.array(which_predictor) != 0

    c = np.zeros(X.shape[1])
    c[idx] = 1 if is_single == 1 else which_predictor[idx]
    des_var = c.dot(np.linalg.inv(X.T.dot(X))).dot(c.T)
    return des_var
```

In [60]:
```
# get a statistical map (a for loop to compute t-value of every voxel)
t_map = f_img_data[:, :, :, 0]  # to store t-values of each voxel (3D map)
for i in range(f_img_data.shape[0]):
    for j in range(f_img_data.shape[1]):
        for k in range(f_img_data.shape[2]):
            voxel_signal = f_img_data[i, j, k, :]
            # start the regression
            if predictor_conv_ds.ndim == 1:
                # Add back a singleton axis (which was removed before downsampling)
                # otherwise stacking will give an error
                predictor_conv_ds = predictor_conv_ds[:, np.newaxis]
            # linear regression
            intercept = np.ones((predictor_conv_ds.size, 1))
            X_conv = np.hstack((intercept, predictor_conv_ds))
            betas_conv = inv(X_conv.T @ X_conv) @ X_conv.T @ voxel_signal
            design_variance_predictor = design_variance(X_conv, which_predictor=1)  # ge
            # get degree of freedom and noise terms
            y_hat = X_conv @ betas_conv
            N = voxel_signal.size
            P = X_conv.shape[1]
            df = (N - P)  # degree of freedom
            sigma_hat = np.sum((voxel_signal - y_hat) ** 2) / df # noise
            #get t-value
            t = betas_conv[1] / np.sqrt(sigma_hat * design_variance_predictor)
            if math.isnan(t):
                t = 0
            # store the t-value in t-map
            t_map[i, j, k] = round(t, 3)
```

In [61]:
```
## convert t-map to NifTi image

# load the data
func = nib.load(fmri_file)

# to save this 3D (ndarry) numpy use this
ni_img = nib.Nifti1Image(t_map, func.affine)
```

In [62]:
```
# Get a cortical mesh
from nilearn import datasets
fsaverage = datasets.fetch_surf_fsaverage()
```

In [63]:
```
# Sample the 3D data around each node of the mesh
from nilearn import surface
texture = surface.vol_to_surf(ni_img, fsaverage.pial_right)
```

plot the result from nilearn import plotting fig = plotting.plot_surf_stat_map(fsaverage.pial_left, texture, cmap = 'PRGn', engine='plotly', threshold="95%", title='Left hemisphere') fig.show()fig = plotting.plot_surf_stat_map(fsaverage.pial_right, texture, cmap = 'PRGn', engine='plotly', threshold="95%", title='Right hemisphere') fig.show()

T-maps Non-Music

Sub-control01

In [64]:
```
# importing the functional MRI data file
fmri_file = 'Dataset/dataset/sub-control01/func/sub-control01_task-nonmusic_run-4_bold.n
f_img = nib.load(fmri_file)
# getting fMRI data
f_img_data = f_img.get_fdata()
```

In [65]:
```
def design_variance(X, which_predictor=1): # X : Numpy Array of shape (N, P)
    is_single = isinstance(which_predictor, int)
    if is_single:
        idx = which_predictor
    else:
        idx = np.array(which_predictor) != 0

    c = np.zeros(X.shape[1])
    c[idx] = 1 if is_single == 1 else which_predictor[idx]
    des_var = c.dot(np.linalg.inv(X.T.dot(X))).dot(c.T)
    return des_var
```

In [66]:
```
# get a statistical map (a for loop to compute t-value of every voxel)
t_map = f_img_data[:, :, :, 0]  # to store t-values of each voxel (3D map)
for i in range(f_img_data.shape[0]):
    for j in range(f_img_data.shape[1]):
        for k in range(f_img_data.shape[2]):
            voxel_signal = f_img_data[i, j, k, :]
            # start the regression
            if predictor_conv_ds.ndim == 1:
                # Add back a singleton axis (which was removed before downsampling)
                # otherwise stacking will give an error
                predictor_conv_ds = predictor_conv_ds[:, np.newaxis]
            # linear regression
            intercept = np.ones((predictor_conv_ds.size, 1))
            X_conv = np.hstack((intercept, predictor_conv_ds))
            betas_conv = inv(X_conv.T @ X_conv) @ X_conv.T @ voxel_signal
            design_variance_predictor = design_variance(X_conv, which_predictor=1)  # ge
            # get degree of freedom and noise terms
            y_hat = X_conv @ betas_conv
            N = voxel_signal.size
            P = X_conv.shape[1]
            df = (N - P)  # degree of freedom
            sigma_hat = np.sum((voxel_signal - y_hat) ** 2) / df # noise
            #get t-value
            t = betas_conv[1] / np.sqrt(sigma_hat * design_variance_predictor)
            if math.isnan(t):
                t = 0
            # store the t-value in t-map
            t_map[i, j, k] = round(t, 3)
```

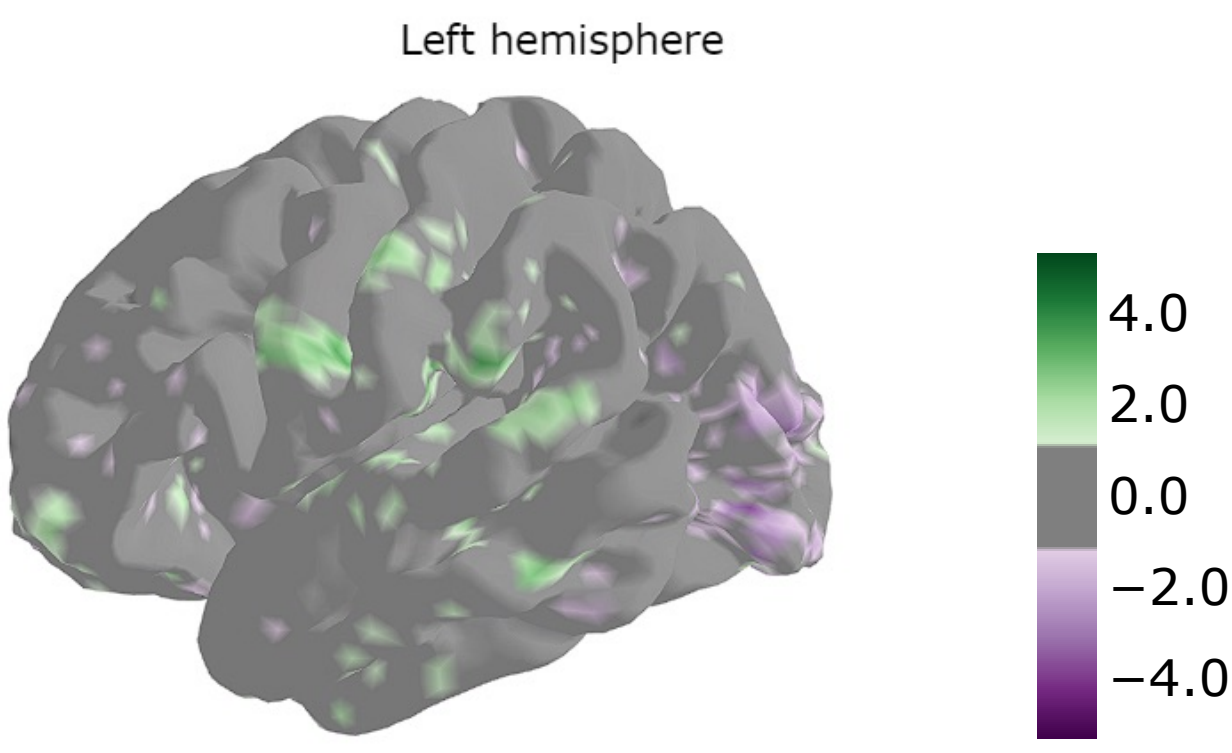
In [67]:
```
## convert t-map to NifTi image

# load the data
func = nib.load(fmri_file)

# to save this 3D (ndarry) numpy use this
ni_img = nib.Nifti1Image(t_map, func.affine)
```

In [68]:
```
# Get a cortical mesh
from nilearn import datasets
fsaverage = datasets.fetch_surf_fsaverage()
```

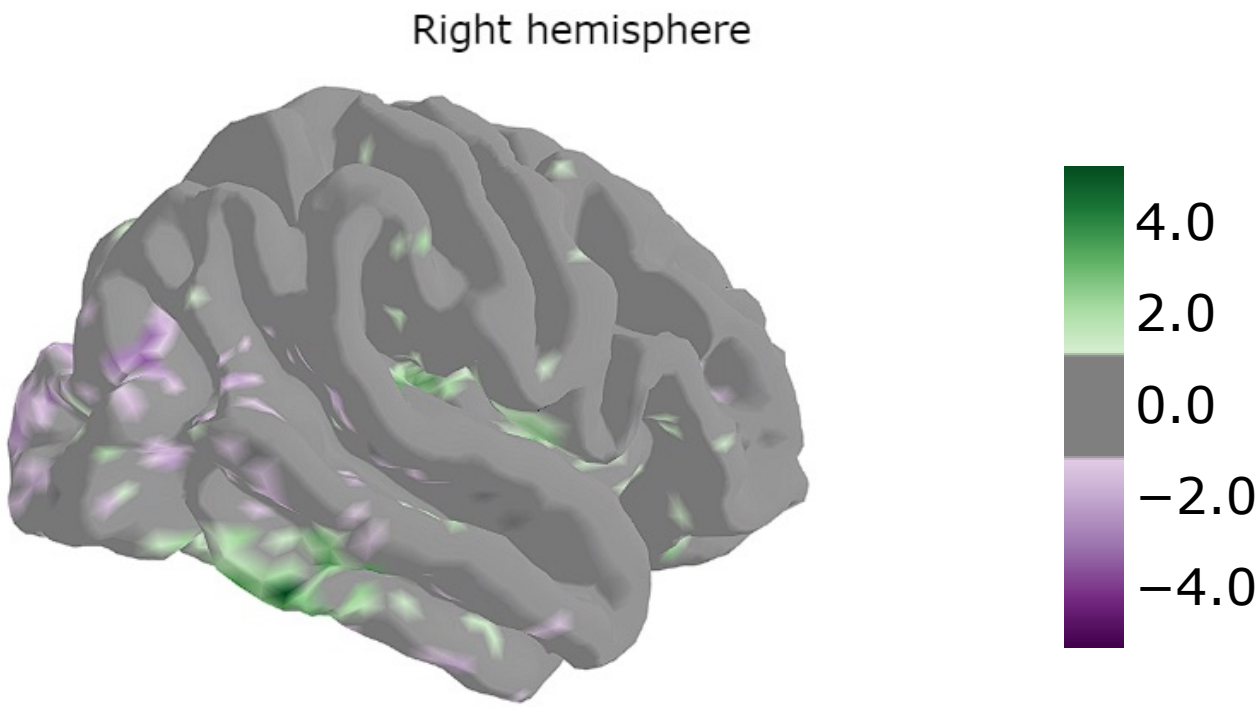
In [69]:
```
# Sample the 3D data around each node of the mesh
```

```
from nilearn import surface
texture = surface.vol_to_surf(ni_img, fsaverage.pial_right)
```

In [70]:
```
# plot the result
from nilearn import plotting

# fig = plotting.plot_surf_stat_map(
#     fsaverage.infl_right, texture, hemi='right',
#     title='Surface right hemisphere', colorbar=True,
#     threshold=0, bg_map=fsaverage.sulc_right,
#     engine='plotly'
# )

fig = plotting.plot_surf_stat_map(fsaverage.pial_left, texture,
                                  cmap = 'PRGn', threshold="90%",
                                  engine='plotly', title='Left hemisphere')
fig.show()
```



In [71]:
```
fig = plotting.plot_surf_stat_map(fsaverage.pial_right, texture,
                                  cmap = 'PRGn', threshold="90%",
                                  engine='plotly', title='Right hemisphere')
fig.show()
```



Sub-mdd01

In [72]:
```
# importing the functional MRI data file
fmri_file = 'Dataset/dataset/sub-mdd01/func/sub-mdd01_task-nonmusic_run-4_bold.nii.gz'
f_img = nib.load(fmri_file)
# getting fMRI data
f_img_data = f_img.get_fdata()
```

In [73]:
```
def design_variance(X, which_predictor=1): # X : Numpy Array of shape (N, P)
    is_single = isinstance(which_predictor, int)
    if is_single:
        idx = which_predictor
    else:
        idx = np.array(which_predictor) != 0

    c = np.zeros(X.shape[1])
    c[idx] = 1 if is_single == 1 else which_predictor[idx]
    des_var = c.dot(np.linalg.inv(X.T.dot(X))).dot(c.T)
    return des_var
```

In [74]:
```
# get a statistical map (a for loop to compute t-value of every voxel)
t_map = f_img_data[:, :, :, 0]  # to store t-values of each voxel (3D map)
for i in range(f_img_data.shape[0]):
    for j in range(f_img_data.shape[1]):
        for k in range(f_img_data.shape[2]):
            voxel_signal = f_img_data[i, j, k, :]
            # start the regression
            if predictor_conv_ds.ndim == 1:
                # Add back a singleton axis (which was removed before downsampling)
                # otherwise stacking will give an error
                predictor_conv_ds = predictor_conv_ds[:, np.newaxis]
            # linear regression
            intercept = np.ones((predictor_conv_ds.size, 1))
            X_conv = np.hstack((intercept, predictor_conv_ds))
            betas_conv = inv(X_conv.T @ X_conv) @ X_conv.T @ voxel_signal
            design_variance_predictor = design_variance(X_conv, which_predictor=1)  # ge
            # get degree of freedom and noise terms
            y_hat = X_conv @ betas_conv
```

```
            N = voxel_signal.size
            P = X_conv.shape[1]
            df = (N - P)  # degree of freedom
            sigma_hat = np.sum((voxel_signal - y_hat) ** 2) / df # noise
            #get t-value
            t = betas_conv[1] / np.sqrt(sigma_hat * design_variance_predictor)
            if math.isnan(t):
                t = 0
            # store the t-value in t-map
            t_map[i, j, k] = round(t, 3)
```

In [75]:
```
## convert t-map to NifTi image

# load the data
func = nib.load(fmri_file)

# to save this 3D (ndarry) numpy use this
ni_img = nib.Nifti1Image(t_map, func.affine)
```

In [76]:
```
# Get a cortical mesh
from nilearn import datasets
fsaverage = datasets.fetch_surf_fsaverage()
```

In [77]:
```
# Sample the 3D data around each node of the mesh
from nilearn import surface
texture = surface.vol_to_surf(ni_img, fsaverage.pial_right)
```

plot the result from nilearn import plotting # fig = plotting.plot_surf_stat_map(# fsaverage.infl_right, texture, hemi='right', # title='Surface right hemisphere', colorbar=True, # threshold=0, bg_map=fsaverage.sulc_right, # engine='plotly' #) fig = plotting.plot_surf_stat_map(fsaverage.pial_left, texture, cmap = 'PRGn', engine='plotly', threshold="90%", title='Left hemisphere') fig.show()fig = plotting.plot_surf_stat_map(fsaverage.pial_right, texture, cmap = 'PRGn', engine='plotly', threshold="90%", title='Right hemisphere') fig.show()

In [ ]:
```

```

Improved GLM

In [78]:
```
# importing the functional MRI data file
fmri_file = 'Dataset\\dataset\\derivatives\\sub-control01\\func\\sub-control01_task-musi
```

In [79]:
```
# check the functional MRI data shape, voxel size, and units
f_img = nib.load(fmri_file)
print(f_img.shape)
print(f_img.header.get_zooms())
print(f_img.header.get_xyzt_units())
```

```
(68, 80, 65, 105)
(2.9, 2.9, 3.0, 3.0)
('mm', 'sec')
```

In [80]:
```
# getting fMRI data
f_img_data = f_img.get_fdata()
print(f_img_data.shape)
```

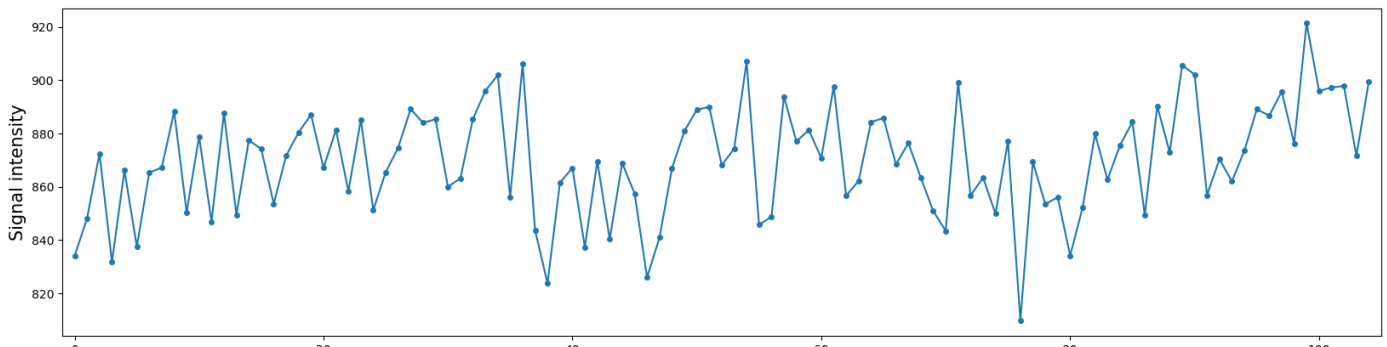
```
(68, 80, 65, 105)
```

In [81]:
```
# extract the time series of the middle voxel
mid_vox_ts = f_img_data[39, 39, 24, :]
print("Voxel timeseries shape: %s" % (mid_vox_ts.shape,))
```

```
Voxel timeseries shape: (105,)
```

In [ ]:
```

```

In [82]:
```
# time-by-time fluctuations in signal intensity by
# plotting the time series directly
plt.figure(figsize=(20, 5))
plt.plot(mid_vox_ts, 'o-', ms=4)
plt.xlim(-1, mid_vox_ts.size)
#plt.ylim(38000, 55000)
plt.ylabel('Signal intensity', fontsize=15)
plt.xlabel('Time (volumes)', fontsize=15)
plt.show()
```



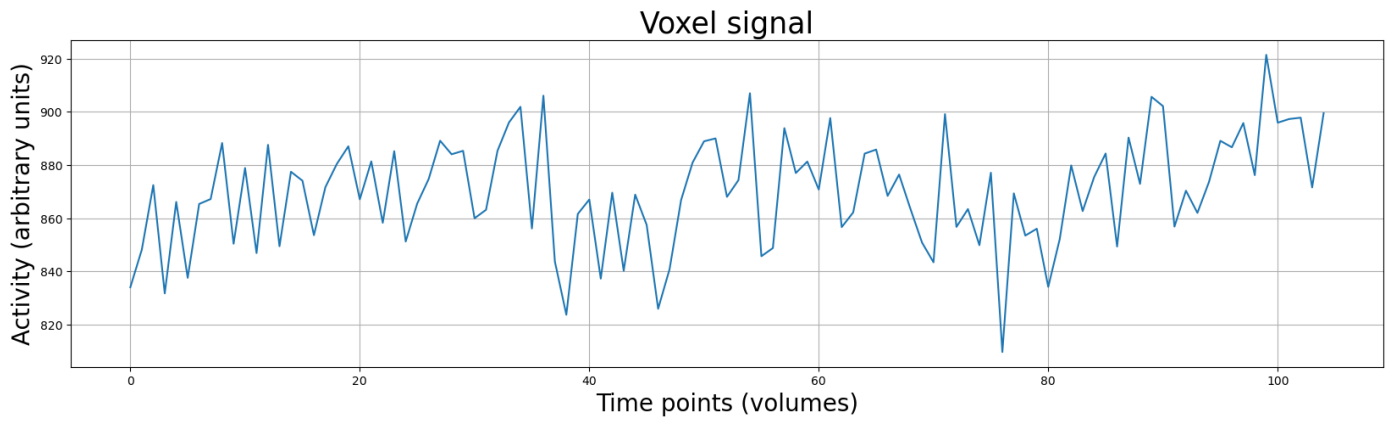
In [83]:
```
# get the events
events_df = pd.read_csv('Dataset/dataset/sub-control01/func/sub-control01_task-music_run
events_df
```

Out[83]:

	onset	duration	trial_type
0	0.0	33.0	tones
1	33.0	3.0	response
2	36.0	31.5	negative_music
3	67.5	3.0	response
4	70.5	31.5	tones
5	102.0	3.0	response
6	105.0	31.5	positive_music
7	136.5	3.0	response
8	139.5	31.5	tones
9	171.0	3.0	response
10	174.0	31.5	negative_music
11	205.5	3.0	response
12	208.5	31.5	tones
13	240.0	3.0	response
14	243.0	31.5	positive_music
15	274.5	3.0	response
16	277.5	31.5	tones
17	309.0	3.0	response

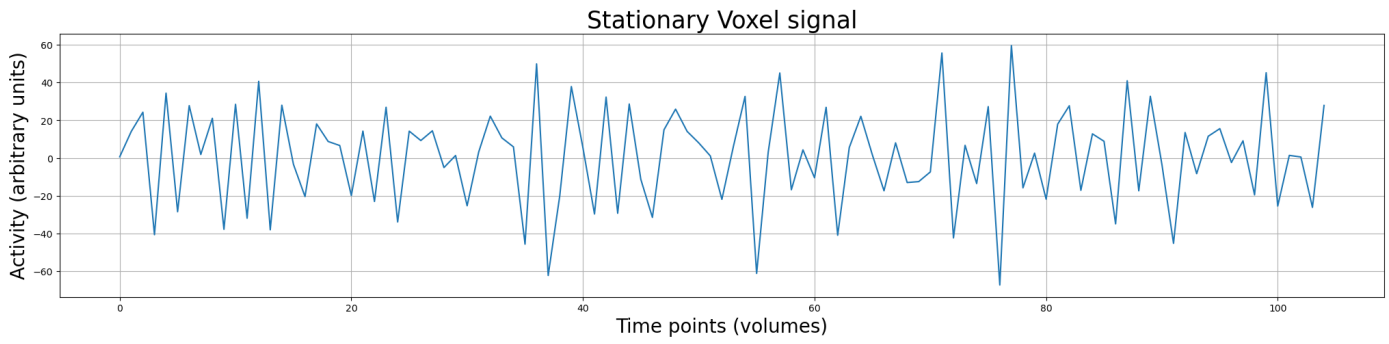
In [84]:
```
# get signal for a voxel
voxel_signal = f_img_data[39, 39, 24, :]
plt.figure(figsize=(20, 5))
plt.plot(voxel_signal)
```

```
plt.xlabel('Time points (volumes)', fontsize=20)
plt.ylabel('Activity (arbitrary units)', fontsize=20)
plt.title('Voxel signal', fontsize=25)
plt.grid()
plt.show()
```



In [85]:
```
# Deal with mean-shift (transform time-series data from non-stationary to stationary)
temp_df = pd.DataFrame(voxel_signal)
# diff() function take the difference of the data point and its previous point
voxel_signal_stationary = temp_df[0].diff()

# the first data point is recorded to 'nan' after transformation
# (because it does have anything before it to subtract with)
# impute that first value with the mean
voxel_signal_stationary = voxel_signal_stationary.fillna(voxel_signal_stationary.mean())
temp_voxel_signal = voxel_signal_stationary.to_numpy()
```

In [86]:
```
# try plotting the transformed
voxel_signal = temp_voxel_signal
plt.figure(figsize=(25, 5))
plt.plot(voxel_signal)
plt.xlabel('Time points (volumes)', fontsize=20)
plt.ylabel('Activity (arbitrary units)', fontsize=20)
# plt.xlim(x_lim)
# plt.ylim(y_lim)
plt.title('Stationary Voxel signal', fontsize=25)
plt.grid()
plt.show()
```



In [87]:
```
## record the response onset in arrays
response_onsets = events_df[events_df['trial_type'] == 'negative_music']
response_onsets = response_onsets.append(events_df[events_df['trial_type'] ==
                                                   'positive_music'])
onsets = response_onsets['onset'].to_numpy()
onsets = onsets.astype(int)
onsets
```

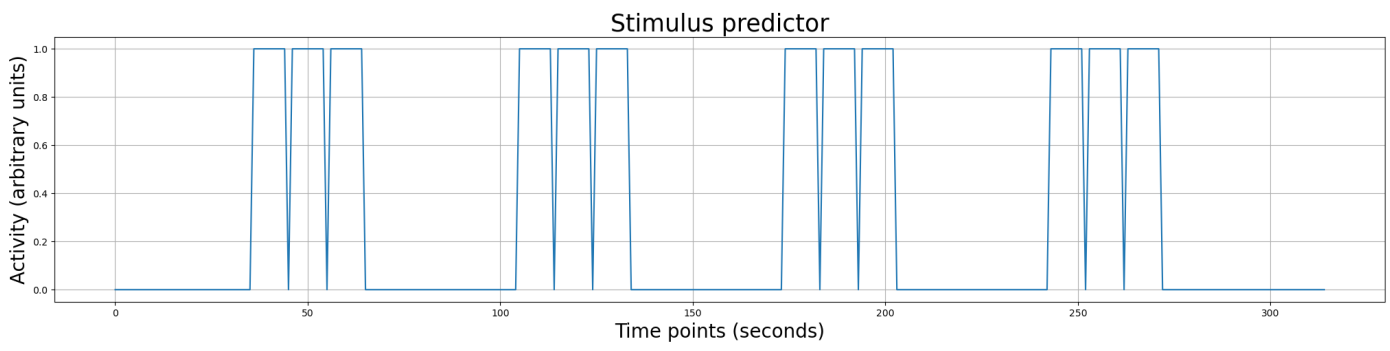
Out[87]:
```
array([ 36, 174, 105, 243])
```

In [88]:
```
## make a range of onsets
onset_range = []
for onset in onsets:
    onset_point = onset
    for i in range(3):
        onset_range = onset_range + list(range(onset_point,onset_point+9))
        onset_point = onset_point + 10
```

In [89]:
```
## convert the onset array to a proper predictor
# (with the same shape as the fMRI signal)
# for congruent condition
predictor = np.zeros(105*3)    # length of experiment is 105 volume with TR = 3s
predictor[onset_range] = 1  # set the predictor at the indices to 1
print("Shape of predictor: %s" % (predictor.shape,))
print("\nContents of predictor array:\n%r" % predictor.T)
```

```
Shape of predictor: (315,)

Contents of predictor array:
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [90]:
```
## visualizing the predictor
plt.figure(figsize=(25, 5))
#plt.plot(predictor_congruent, marker='o')
plt.plot(predictor, c='tab:blue')
plt.xlabel('Time points (seconds)', fontsize=20)
plt.ylabel('Activity (arbitrary units)', fontsize=20)
#plt.xlim(0, 300)
#plt.ylim(-.5, 1.5)
plt.title('Stimulus predictor', fontsize=25)
plt.grid()
plt.show()
```

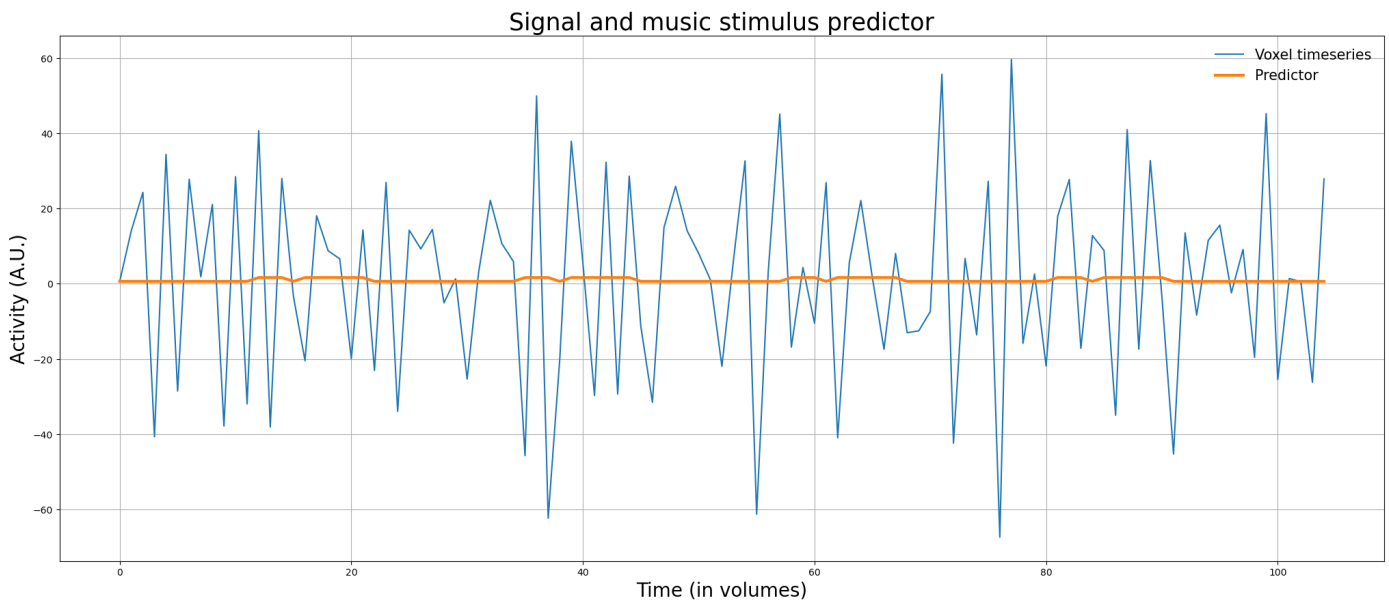


In [91]:
```
## since the voxel in fMRI was measured in volume with TR of 3 seconds,
## we have to down sampling the predictor to fit the same unit of time.
from scipy.interpolate import interp1d
```

```
original_scale = np.arange(0, 105*3, 1) # from 0 to 105*3 seconds
print("Original scale has %i datapoints (0-315, in seconds)" %
      original_scale.size)
resampler = interp1d(original_scale, predictor)
desired_scale = np.arange(0, 105*3, 3)
print("Desired scale has %i datapoints (0, 3, 6, ... 105, in volumes)" %
      desired_scale.size)
predictor_ds = resampler(desired_scale)
print("Downsampled predictor has %i datapoints (in volumes)" %
      predictor_ds.size)
print(predictor_ds)
```

```
Original scale has 315 datapoints (0-315, in seconds)
Desired scale has 105 datapoints (0, 3, 6, ... 105, in volumes)
Downsampled predictor has 105 datapoints (in volumes)
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

In [92]:
```
## inspecting the predictor and the actual signal before moving on to linear regression
plt.figure(figsize=(25, 10))
plt.plot(voxel_signal)
plt.plot(predictor_ds + voxel_signal.mean(), lw=3)
plt.xlabel('Time (in volumes)', fontsize=20)
plt.ylabel('Activity (A.U.)', fontsize=20)
plt.legend(['Voxel timeseries', 'Predictor'], fontsize=15,
           loc='upper right', frameon=False)
plt.title("Signal and music stimulus predictor", fontsize=25)
plt.grid()
plt.show()
```



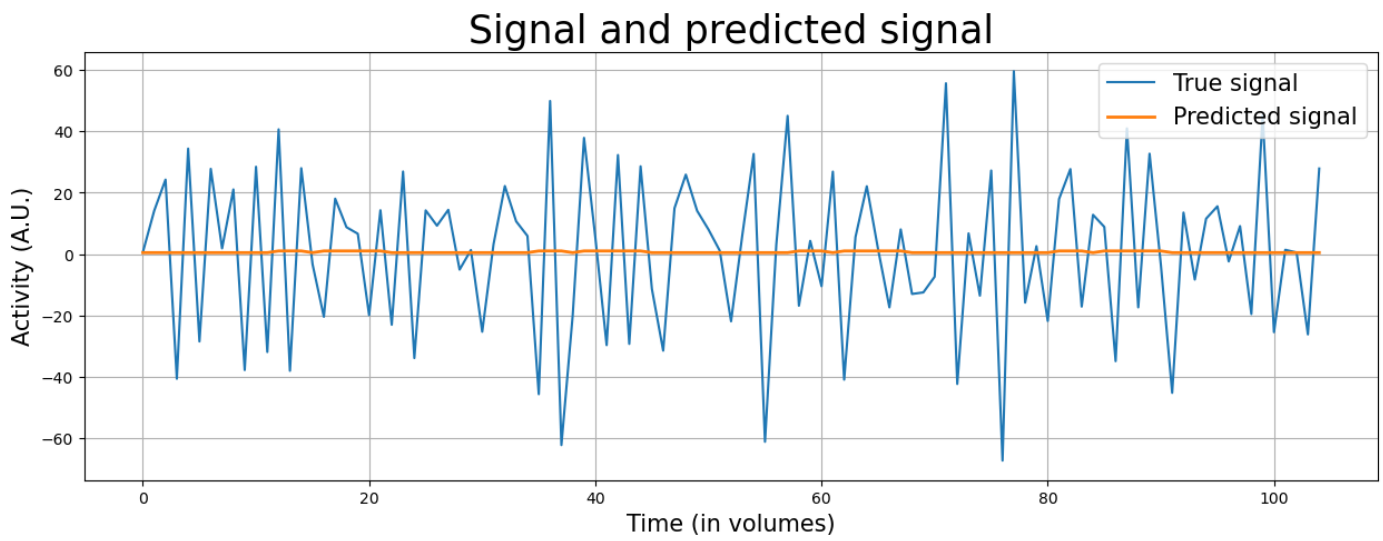
In [93]:
```
# Regression
if predictor_ds.ndim == 1: # This adds a singleton dimension, such that you can call np.
    predictor_ds = predictor_ds[:, np.newaxis]

icept = np.ones((predictor_ds.size, 1))
X_simple = np.hstack((icept, predictor_ds))
betas_simple = inv(X_simple.T @ X_simple) @ X_simple.T @ voxel_signal
y_hat = X_simple[:, 0] * betas_simple[0] + X_simple[:, 1] * betas_simple[1]
print(betas_simple)
numerator = np.sum((voxel_signal - y_hat) ** 2)
denominator = np.sum((voxel_signal - np.mean(voxel_signal)) ** 2)
```

```
r_squared = 1 - numerator / denominator
print('The R² value is: %.3f' % r_squared)
```

```
[0.44040361 0.54975589]
The R² value is: 0.000
```

In [94]:
```
# function for plotting predicted signal vs. actual signal
def plot_signal_and_predicted_signal(y, X, x_lim=None, y_lim=None):
    """ Plots a signal and its GLM prediction. """
    des = np.hstack((np.ones((y.size, 1)), X))
    betas_simple = np.linalg.lstsq(des, y, rcond=None)[0]
    plt.figure(figsize=(15, 5))
    plt.plot(y)
    plt.plot(des @ betas_simple, lw=2)
    plt.xlabel('Time (in volumes)', fontsize=15)
    plt.ylabel('Activity (A.U.)', fontsize=15)
    plt.legend(['True signal', 'Predicted signal'],
               loc='upper right', fontsize=15)
    plt.title("Signal and predicted signal", fontsize=25)
    plt.grid()
    plt.show()
```

In [95]:
```
# plotting predicted signal vs. actual signal
plot_signal_and_predicted_signal(voxel_signal, predictor_ds)
```

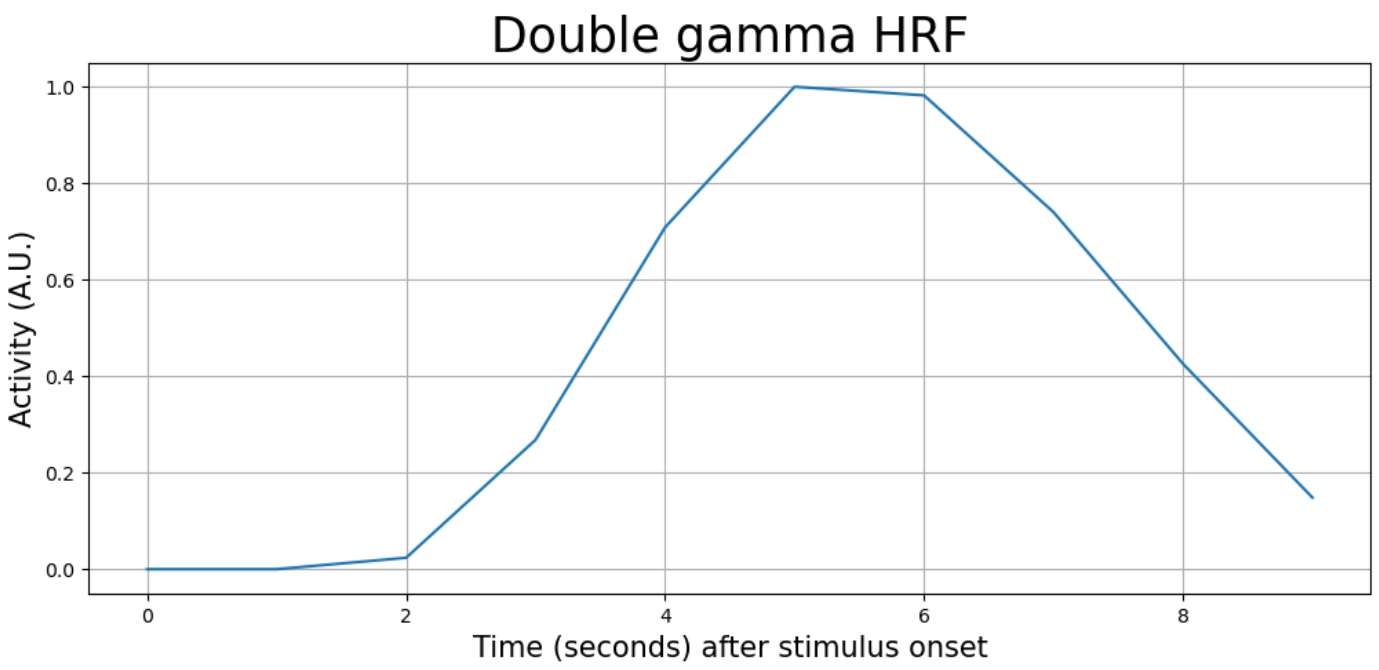


In [96]:
```
# Now, model with BOLD response function
# install nilearn package
# import sys
# !{sys.executable} -m pip install nilearn

# use glover hrf from nilearn
from nilearn.glm.first_level.hemodynamic_models import glover_hrf

# creating the canonical HFR
TR = 3
osf = 3
length_hrf = 10 # sec # have to shape it according to our data
# (let's say to match the ITI of the experiment)
canonical_hrf = glover_hrf(tr=TR, oversampling=osf,
                           time_length=length_hrf, onset=0)
canonical_hrf /= canonical_hrf.max()
print("Size of canonical hrf variable: %i" % canonical_hrf.size)
```

```
Size of canonical hrf variable: 10
```

In [97]:
```
# visualizing the canonical HRF
t = np.arange(0, canonical_hrf.size)
```

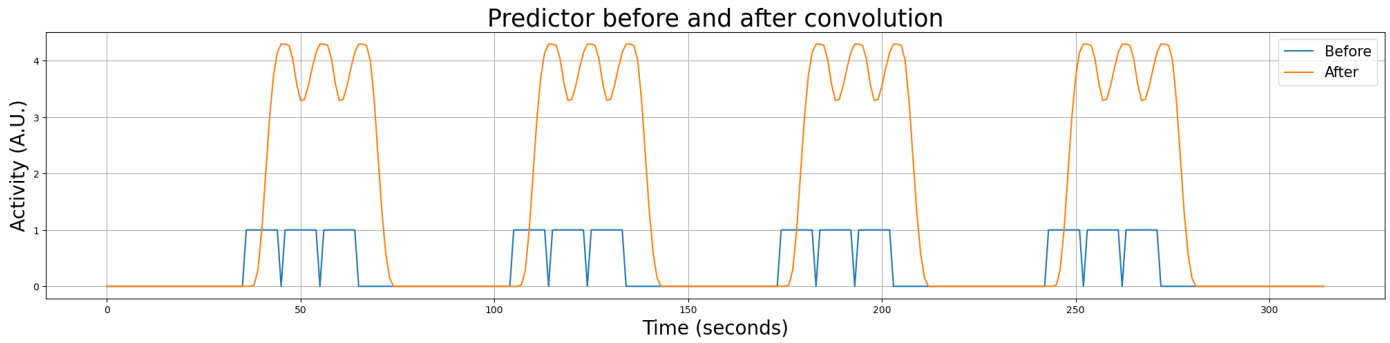
```
plt.figure(figsize=(12, 5))
plt.plot(t,canonical_hrf)
plt.xlabel('Time (seconds) after stimulus onset', fontsize=15)
plt.ylabel('Activity (A.U.)', fontsize=15)
plt.title('Double gamma HRF', fontsize=25)
plt.grid()
plt.show()
```



In [98]:
```
# convolve the predictor with the HRF
predictor_conv = np.convolve(predictor.squeeze(), canonical_hrf)
print("The shape of the convolved predictor after convolution: %s" % (predictor_conv.sha

# After convolution, we also neem to "trim" off some excess
# values from the convolved signal
predictor_conv = predictor_conv[:predictor.size]
print("After trimming, the shape is: %s" % (predictor_conv.shape,))

# And we have to add a new axis again to go from shape (N,) to (N, 1),
# which is important for stacking the intercept later
predictor_conv = predictor_conv[:, np.newaxis]
print("Shape after adding the new axis: %s" % (predictor_conv.shape,))
```

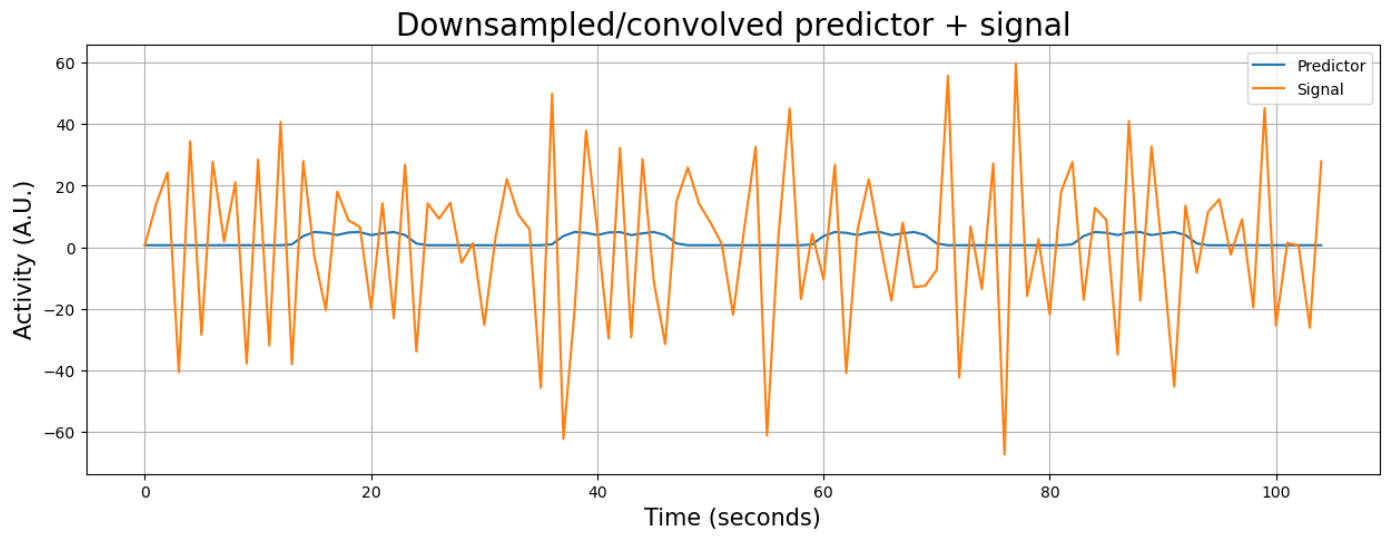
```
The shape of the convolved predictor after convolution: (324,)
After trimming, the shape is: (315,)
Shape after adding the new axis: (315, 1)
```

In [99]:
```
# visualizing the predictor before and after the convolution
# congruent
plt.figure(figsize=(25, 5))
plt.plot(predictor)
plt.plot(predictor_conv)
#plt.xlim(-1, 800)
plt.title("Predictor before and after convolution", fontsize=25)
plt.xlabel("Time (seconds)", fontsize=20)
plt.ylabel("Activity (A.U.)", fontsize=20)
plt.legend(['Before', 'After'], loc='upper right', fontsize=15)
plt.grid()
plt.show()
```



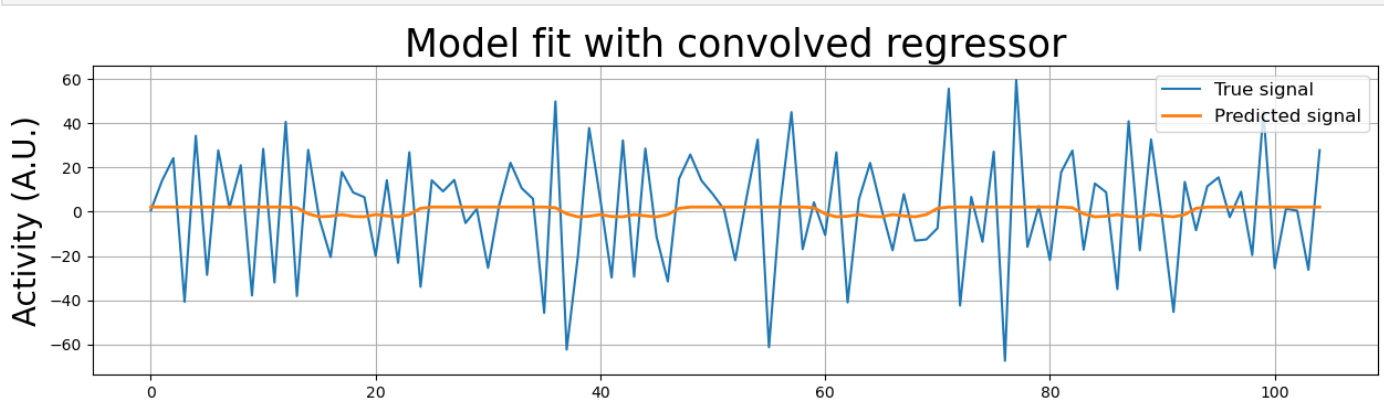
In [100…

```
# scale the convolved predictor back to the scale of volume
original_scale = np.arange(0, 105*3, 1) # from 0 to 105*3 seconds
resampler = interp1d(original_scale, np.squeeze(predictor_conv))
desired_scale = np.arange(0, 105*3, 3)
predictor_conv_ds = resampler(desired_scale)
# x_lim, y_lim = (0, 400), (990, 1020)
plt.figure(figsize=(15, 5))
plt.plot(predictor_conv_ds + voxel_signal.mean())
plt.plot(voxel_signal)
plt.grid()
plt.title('Downsampled/convolved predictor + signal', fontsize=20)
plt.ylabel('Activity (A.U.)', fontsize=15)
plt.xlabel('Time (seconds)', fontsize=15)
plt.legend(['Predictor', 'Signal'])
# plt.xlim(x_lim)
plt.show()
```



In [101…

```
# Now, fitting convolved predictor in GLM
if predictor_conv_ds.ndim == 1:
    # Add back a singleton axis (which was removed before downsampling)
    # otherwise stacking will give an error
    predictor_conv_ds = predictor_conv_ds[:, np.newaxis]

intercept = np.ones((predictor_conv_ds.size, 1))
X_conv = np.hstack((intercept, predictor_conv_ds))
betas_conv = inv(X_conv.T @ X_conv) @ X_conv.T @ voxel_signal
plt.figure(figsize=(15, 8))
plt.subplot(2, 1, 1)
plt.plot(voxel_signal)
plt.plot(X_conv @ betas_conv, lw=2)
# plt.xlim(x_lim)
plt.ylabel("Activity (A.U.)", fontsize=20)
plt.title("Model fit with convolved regressor", fontsize=25)
plt.legend(['True signal', 'Predicted signal'], fontsize=12,
           loc='upper right')
plt.grid()
```



In [102…
```
# Evaluate the model
from numpy.linalg import lstsq # numpy implementation of OLS, because we're lazy
y_hat_conv = X_conv @ betas_conv
y_hat_orig = X_simple @ lstsq(X_simple, voxel_signal, rcond=None)[0]
MSE_conv = ((y_hat_conv - voxel_signal) ** 2).mean()
MSE_orig = ((y_hat_orig - voxel_signal) ** 2).mean()
print("MSE of model with convolution is %.3f while the MSE of the model without convolut
 (MSE_conv, MSE_orig))
R2_conv = 1 - (np.sum((voxel_signal - y_hat_conv) ** 2) / np.sum((voxel_signal - voxel_s
R2_orig = 1 - (np.sum((voxel_signal - y_hat_orig) ** 2) / np.sum((voxel_signal - voxel_s
print("R-squared of model with convolution is %.5f and without convolution it is %.5f."
 (R2_conv, R2_orig))
```

```
MSE of model with convolution is 701.970 while the MSE of the model without convolution
is 705.477.
R-squared of model with convolution is 0.00507 and without convolution it is 0.00010.
```

As can be seen here, the R2 is much better than the previous GLM

fmriprep preproc Data

In [103…
```
# importing the functional MRI data file
fmri_file = 'Dataset\\dataset\\derivatives\\sub-control01\\func\\sub-control01_task-musi
f_img = nib.load(fmri_file)
# getting fMRI data
f_img_data = f_img.get_fdata()
```

In [104…
```
def design_variance(X, which_predictor=1): # X : Numpy Array of shape (N, P)
    is_single = isinstance(which_predictor, int)
    if is_single:
        idx = which_predictor
    else:
        idx = np.array(which_predictor) != 0
    c = np.zeros(X.shape[1])
    c[idx] = 1 if is_single == 1 else which_predictor[idx]
    des_var = c.dot(np.linalg.inv(X.T.dot(X))).dot(c.T)
    return des_var
```

In [105…
```
# get a statistical map (a for loop to compute t-value of every voxel)
t_map = f_img_data[:, :, :, 0]  # to store t-values of each voxel (3D map)
for i in range(f_img_data.shape[0]):
    for j in range(f_img_data.shape[1]):
        for k in range(f_img_data.shape[2]):
            voxel_signal = f_img_data[i, j, k, :]
            # start the regression
            if predictor_conv_ds.ndim == 1:
                # Add back a singleton axis (which was removed before downsampling)
                # otherwise stacking will give an error
```

```
            predictor_conv_ds = predictor_conv_ds[:, np.newaxis]
        # linear regression
        intercept = np.ones((predictor_conv_ds.size, 1))
        X_conv = np.hstack((intercept, predictor_conv_ds))
        betas_conv = inv(X_conv.T @ X_conv) @ X_conv.T @ voxel_signal
        design_variance_predictor = design_variance(X_conv, which_predictor=1)  # ge
        # get degree of freedom and noise terms
        y_hat = X_conv @ betas_conv
        N = voxel_signal.size
        P = X_conv.shape[1]
        df = (N - P)  # degree of freedom
        sigma_hat = np.sum((voxel_signal - y_hat) ** 2) / df # noise
        #get t-value
        t = betas_conv[1] / np.sqrt(sigma_hat * design_variance_predictor)
        if math.isnan(t):
            t = 0
        # store the t-value in t-map
        t_map[i, j, k] = round(t, 3)
```

In [106…

```
t_map
```

Out[106]:

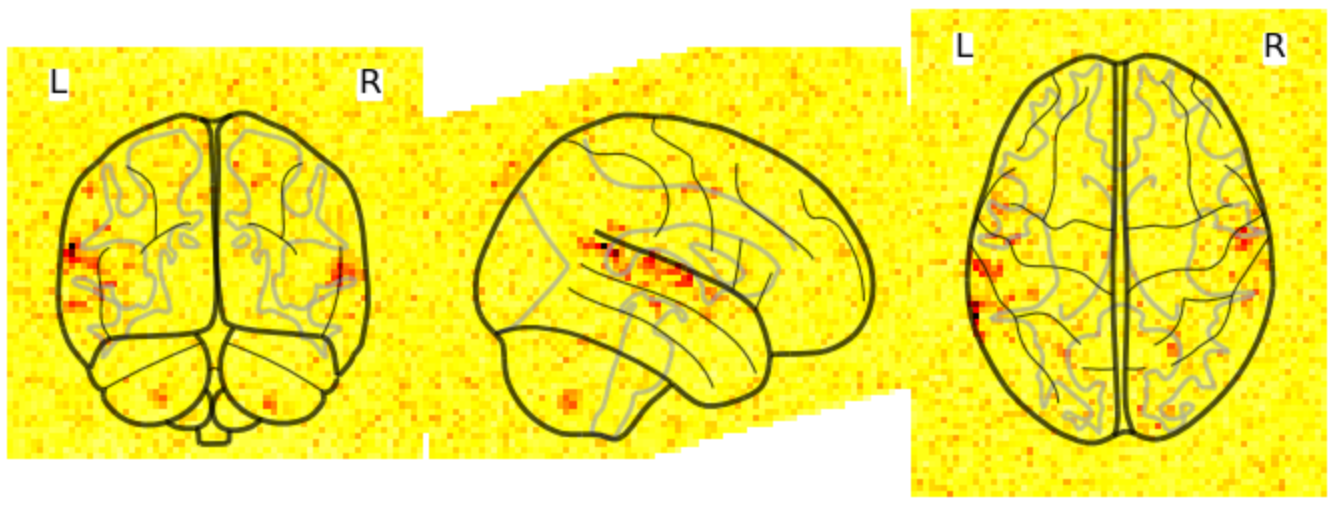
```
array([[[ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ],
        [-1.341, -1.853, -0.658, ...,  0.   ,  0.   ,  0.   ],
        [-2.256, -2.508, -0.397, ...,  0.   ,  0.   ,  0.   ],
        ...,
        [ 0.   ,  0.   ,  0.   , ..., -0.836,  0.   ,  0.   ],
        [ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ],
        [ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ]],

       [[ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ],
        [ 0.35 ,  1.642,  0.8  , ...,  0.   ,  0.   ,  0.   ],
        [-0.806,  0.211,  1.48 , ...,  0.   ,  0.   ,  0.   ],
        ...,
        [ 0.   ,  0.   ,  0.   , ..., -1.389,  0.   ,  0.   ],
        [ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ],
        [ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ]],

       [[ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ],
        [ 2.228,  0.266, -0.43 , ...,  0.   ,  0.   ,  0.   ],
        [ 1.319,  0.221, -0.547, ...,  0.   ,  0.   ,  0.   ],
        ...,
        [ 0.   ,  0.   ,  0.   , ..., -0.052,  0.   ,  0.   ],
        [ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ],
        [ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ]],

       ...,

       [[ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ],
        [ 0.   ,  0.   , -1.182, ...,  0.   ,  0.   ,  0.   ],
        [ 0.703, -0.892,  0.294, ...,  0.   ,  0.   ,  0.   ],
        ...,
        [ 0.   ,  0.   ,  0.   , ...,  1.142,  0.119, -1.105],
        [ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ],
        [ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ]],

       [[ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ],
        [ 0.   ,  0.   , -1.649, ...,  0.   ,  0.   ,  0.   ],
        [ 2.752,  0.499, -0.041, ...,  0.   ,  0.   ,  0.   ],
        ...,
        [ 0.   ,  0.   ,  0.   , ...,  0.052,  0.274,  0.914],
        [ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ],
        [ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ]],

       [[ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ],
        [ 0.   ,  0.   , -2.886, ...,  0.   ,  0.   ,  0.   ],
```

```
[ 0.961,  0.038, -0.851, ...,  0.   ,  0.   ,  0.   ],
...,
[ 0.   ,  0.   ,  0.   , ...,  0.967,  0.071, -0.136],
[ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ],
[ 0.   ,  0.   ,  0.   , ...,  0.   ,  0.   ,  0.   ]])
```

In [107…
```
## convert t-map to NifTi image

# load the data
func = nib.load(fmri_file)

# to save this 3D (ndarry) numpy use this
ni_img = nib.Nifti1Image(t_map, func.affine)
```

In [108…
```
from nilearn import plotting
plotting.plot_glass_brain(ni_img, threshold=0.5)
```

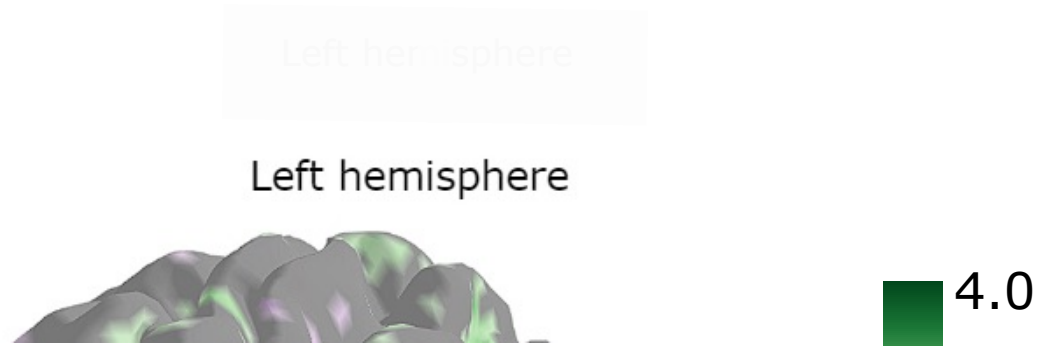
Out[108]:
```
<nilearn.plotting.displays._projectors.OrthoProjector at 0x2c21bafaca0>
```



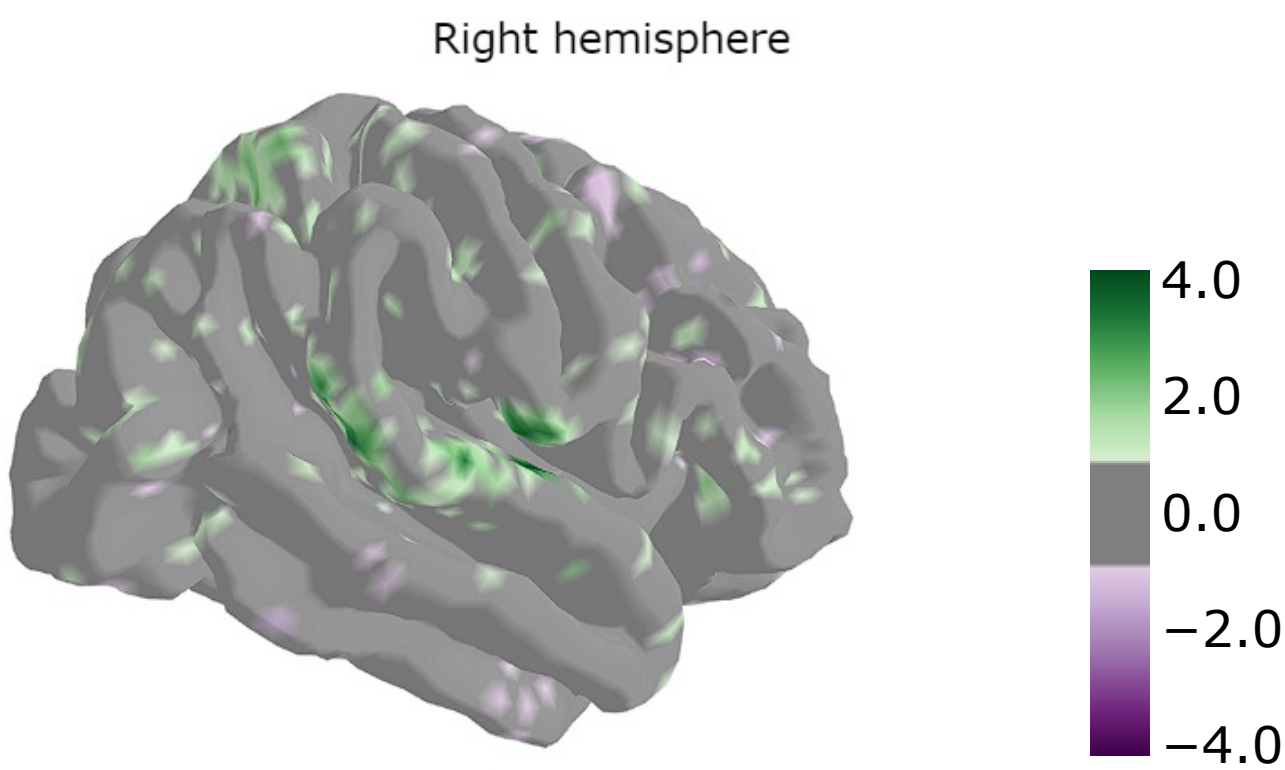
In [109…
```
# Get a cortical mesh
from nilearn import datasets
fsaverage = datasets.fetch_surf_fsaverage()
```

In [110…
```
# Sample the 3D data around each node of the mesh
from nilearn import surface
texture = surface.vol_to_surf(ni_img, fsaverage.pial_right)
```

In [111…
```
# plot the result
from nilearn import plotting

fig = plotting.plot_surf_stat_map(fsaverage.pial_left, texture,
                                  cmap = 'PRGn', threshold="90%",
                                  engine='plotly',  title='Left hemisphere')
fig.show()
```



In [112…
```
fig = plotting.plot_surf_stat_map(fsaverage.pial_right, texture,
                                  cmap = 'PRGn', threshold="90%",
                                  engine='plotly',  title='Right hemisphere')
fig.show()
```



In [ ]:

Functional Connectivity Analysis

In [113…
```
# install pybids module
# !{sys.executable} -m pip install pybids
import bids
```

In [114…
```
layout = bids.BIDSLayout('Dataset\dataset\derivatives',
                         config=['bids','derivatives'])
```

In [115…
```
layout.get_subjects()
```

Out[115]:
```
['mdd02', 'mdd01', 'control02', 'control01']
```

In [116…
```
layout.get_tasks()
```

Out[116]:
```
['music', 'nonmusic']
```

Cleaning Confounders

In [117…
```
from nilearn import image as nimg
from nilearn import plotting as nplot
```

In [118…
```
# Setting up Motion Estimates
sub = 'control01'
fmriprep_dir = 'Dataset\dataset\derivatives'
layout = bids.BIDSLayout(fmriprep_dir,validate=False,
                         config=['bids','derivatives'])
```

In [119…
```
func_files = layout.get(subject=sub,
                        datatype='func', task='music',
                        desc='preproc',
                        space='MNI152NLin2009cAsym',
                        extension='nii.gz',
                        return_type='file')

mask_files = layout.get(subject=sub,
                        datatype='func', task='music',
                        desc='brain',
                        suffix='mask',
                        space='MNI152NLin2009cAsym',
                        extension="nii.gz",
                        return_type='file')

confound_files = layout.get(subject=sub,
                            datatype='func', task='music',
                            desc='confounds',
                            extension="tsv",
                            return_type='file')
```

In [120…
```
func_file = func_files[0]
mask_file = mask_files[0]
confound_file = confound_files[0]
```

In [121…
```
# read in the confounds.tsv file
confound_df = pd.read_csv(confound_file, delimiter='\t')
confound_df.head()
```

Out[121]:

	global_signal	global_signal_derivative1	global_signal_derivative1_power2	global_signal_power2	csf	cs

0	917.228117	NaN	NaN	841307.418997	1353.110302
1	918.086491	0.858374	0.736806	842882.805225	1347.072759
2	918.275295	0.188804	0.035647	843229.516815	1350.422078
3	916.990980	-1.284314	1.649463	840872.458120	1349.577096
4	916.229643	-0.761338	0.579635	839476.758271	1346.257350

5 rows × 145 columns

The Yeo 2011 Pre-processing schema Confound regressors 6 motion parameters (trans_x, trans_y, trans_z, rot_x, rot_y, rot_z) Global signal (global_signal) Cerebral spinal fluid signal (csf) White matter signal (white_matter) This is a total of 9 base confound regressor variables. Finally we add temporal derivatives of each of these signals as well (1 temporal derivative for each), the result is 18 confound regressors.

In [122…
```
# Setting up Confound variables for regression
# Computing temporal derivatives for confound variables
# Select confounds
confound_vars = ['trans_x','trans_y','trans_z', 'rot_x','rot_y','rot_z',
                 'global_signal', 'csf', 'white_matter']
```

In [123…
```
# pick the derivatives for our confound_vars
# Get derivative column names
derivative_columns = ['{}_derivative1'.format(c) for c
                      in confound_vars]

print(derivative_columns)
```

```
['trans_x_derivative1', 'trans_y_derivative1', 'trans_z_derivative1', 'rot_x_derivative
1', 'rot_y_derivative1', 'rot_z_derivative1', 'global_signal_derivative1', 'csf_derivati
ve1', 'white_matter_derivative1']
```

In [124…
```
# join two lists together
final_confounds = confound_vars + derivative_columns
print(final_confounds)
```

```
['trans_x', 'trans_y', 'trans_z', 'rot_x', 'rot_y', 'rot_z', 'global_signal', 'csf', 'wh
ite_matter', 'trans_x_derivative1', 'trans_y_derivative1', 'trans_z_derivative1', 'rot_x
_derivative1', 'rot_y_derivative1', 'rot_z_derivative1', 'global_signal_derivative1', 'c
sf_derivative1', 'white_matter_derivative1']
```

In [125…
```
confound_df = confound_df[final_confounds]
confound_df.head()
```

Out[125]:

	trans_x	trans_y	trans_z	rot_x	rot_y	rot_z	global_signal	csf	white_matter	tran
0	0.011545	-0.066932	-0.040301	0.001288	-6.168340e-05	0.000530	917.228117	1353.110302	880.207453	
1	-0.000138	-0.047216	-0.040344	0.000936	-2.465900e-04	0.000514	918.086491	1347.072759	880.676478	
2	0.015036	-0.032261	-0.025076	0.001284	-9.896660e-05	0.000448	918.275295	1350.422078	880.836031	
3	-0.000120	-0.043132	-0.046193	0.001309	6.751990e-23	0.000311	916.990980	1349.577096	880.936306	
4	-0.000114	-0.041067	-0.069035	0.001440	5.106010e-05	0.000500	916.229643	1346.257350	880.574347	

In [126…
```
# Dummy TR Drop
# load in our data and check the shape
raw_func_img = nimg.load_img(func_file)
raw_func_img.shape
```

Out[126]:
```
(68, 80, 65, 105)
```

In [127…
```
# drop first 4 timepoints
func_img = raw_func_img.slicer[:,:,:,4:]
func_img.shape
```

Out[127]:
```
(68, 80, 65, 101)
```

In [128…
```
#Drop confound dummy TRs
drop_confound_df = confound_df.loc[4:]
print(drop_confound_df.shape) #number of rows should match that of the functional image
# drop_confound_df.head()
```

```
(101, 18)
```

In [129…
```
# Applying confound regression -- clean our data of our selected confound variables
confounds_matrix = drop_confound_df.values

#Confirm matrix size is correct
confounds_matrix.shape
```

Out[129]:
```
(101, 18)
```

In [130…
```
# Set some constants
high_pass= 0.009
low_pass = 0.08
t_r = 3

# Clean data
clean_img = nimg.clean_img(func_img, confounds=confounds_matrix,
                           detrend=True, standardize=True,
                           low_pass=low_pass, high_pass=high_pass, t_r=t_r)
```

Applying Parcellations to Data

In [131…
```
# Retrieving the Atlas
# using a set of parcellation from Yeo et al. 2011.
from nilearn import datasets
parcel_dir = 'resources/rois/'
atlas_yeo_2011 = datasets.fetch_atlas_yeo_2011(parcel_dir)
```

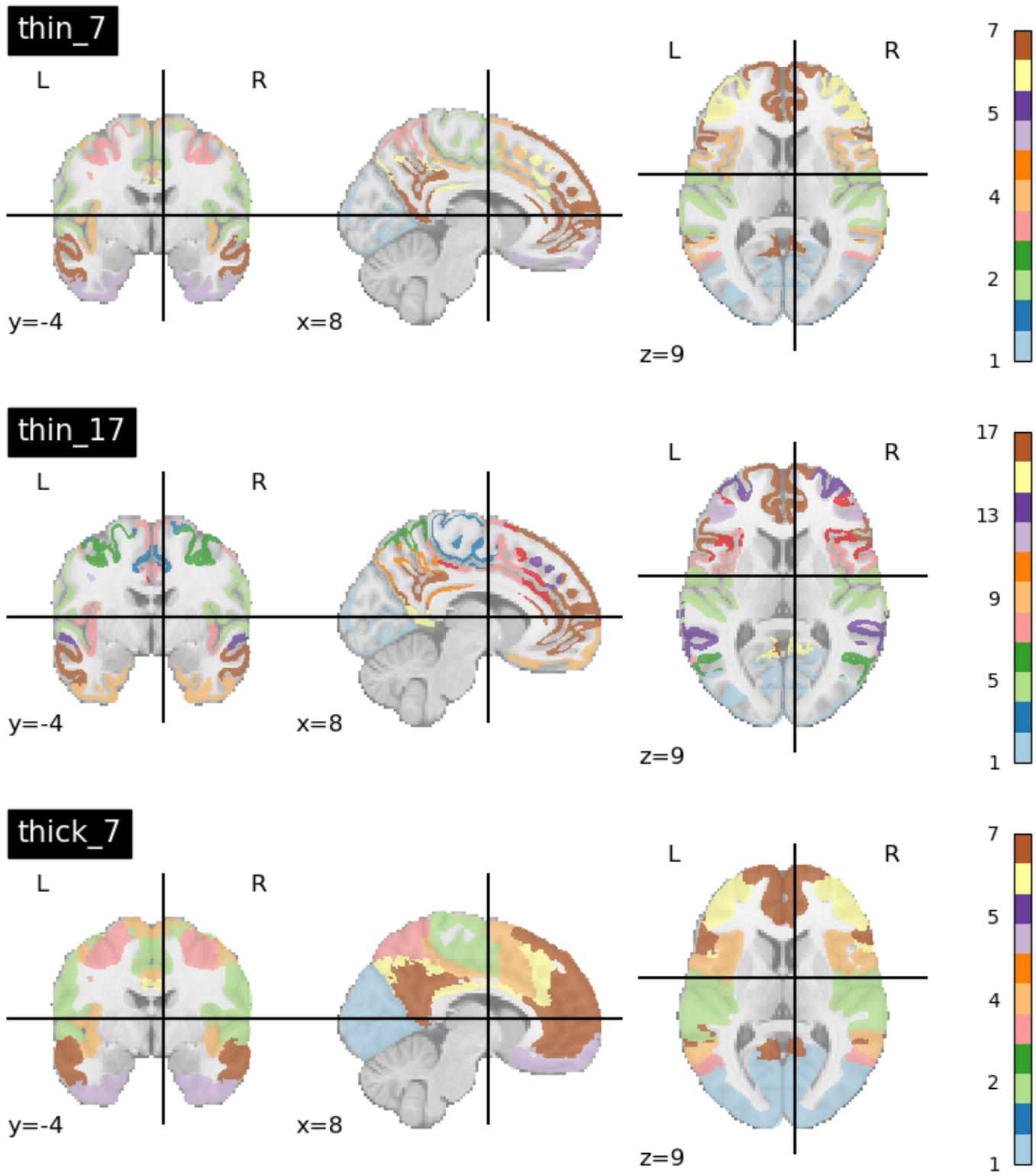
In [132…
```
atlas_yeo_2011.keys()
```

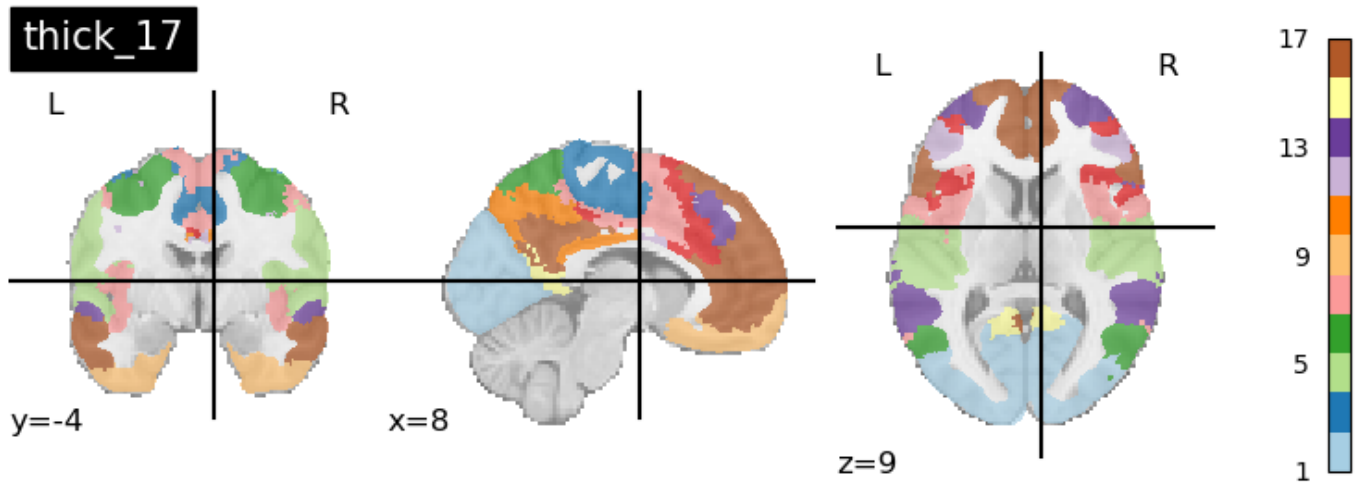
Out[132]:
```
dict_keys(['description', 'thin_7', 'thick_7', 'thin_17', 'thick_17', 'colors_7', 'color
s_17', 'anat'])
```

In [133…
```
# checking Yeo Atlas
#Define where to slice the image
cut_coords = (8, -4, 9)
#Show a colorbar
colorbar=True
#Color scheme to show when viewing image
cmap='Paired'
```

```
#Plot all parcellation schemas referred to by atlas_yeo_2011
nplot.plot_roi(atlas_yeo_2011['thin_7'], cut_coords=cut_coords,
               colorbar=colorbar, cmap=cmap, title='thin_7')
nplot.plot_roi(atlas_yeo_2011['thin_17'], cut_coords=cut_coords,
               colorbar=colorbar, cmap=cmap, title='thin_17')
nplot.plot_roi(atlas_yeo_2011['thick_7'], cut_coords=cut_coords,
               colorbar=colorbar, cmap=cmap, title='thick_7')
nplot.plot_roi(atlas_yeo_2011['thick_17'], cut_coords=cut_coords,
               colorbar=colorbar, cmap=cmap, title='thick_17')
```

Out[133]: <nilearn.plotting.displays._slicers.OrthoSlicer at 0x2c21ec50400>



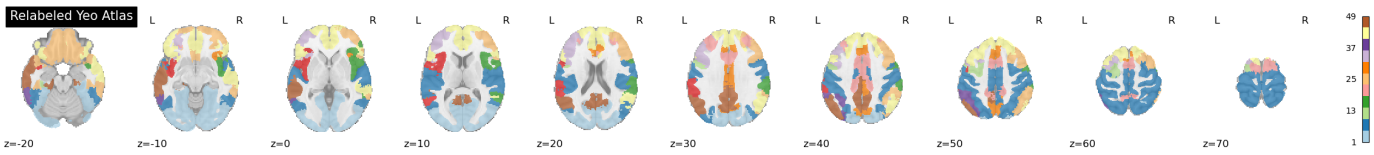


thick_7 variation which includes the following networks: - Visual - Somatosensory - Dorsal Attention - Ventral Attention - Limbic - Frontoparietal - Default The parcel areas labelled with 0 are background voxels not associated with a particular network

In [134…
```
# get thick_7 atlas
atlas_yeo = atlas_yeo_2011['thick_7']
```

In [135…
```
# Spatial Separation of Network
from nilearn.regions import connected_label_regions
region_labels = connected_label_regions(atlas_yeo)
nplot.plot_roi(region_labels, cut_coords=(-20,-10,0,10,20,30,40,50,60,70),
               display_mode='z',colorbar=True,cmap='Paired',
               title='Relabeled Yeo Atlas')
```

Out[135]:
```
<nilearn.plotting.displays._slicers.ZSlicer at 0x2c21f5fc550>
```



In [136…
```
# Resampling the Atlas
# store the separated version of the atlas into a NIFTI file to work with it later
region_labels.to_filename('resources/rois/yeo_2011/Yeo_JNeurophysiol11_MNI152/relabeled_
```

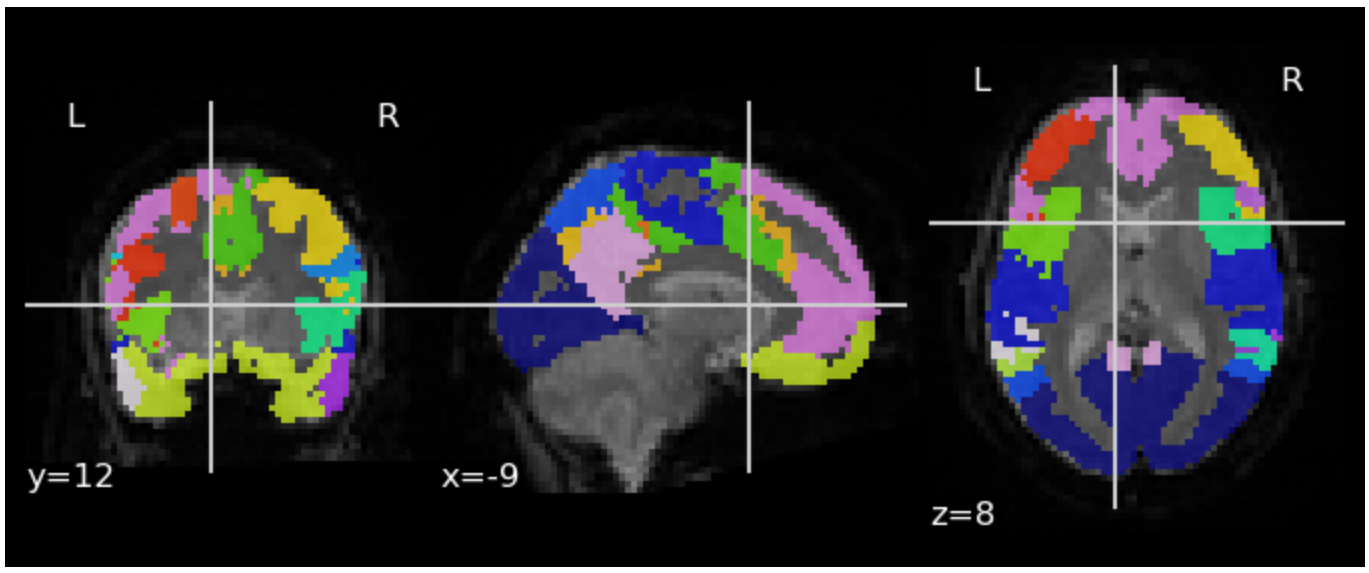
In [137…
```
func_img = nib.load(func_file)
```

In [138…
```
# Print dimensions of functional image and atlas image

print("Size of functional image:", func_img.shape)
print("Size of atlas image:", region_labels.shape)


resampled_yeo = nimg.resample_to_img(region_labels, func_img,
                                     interpolation = 'nearest')
```

```
Size of functional image: (68, 80, 65, 105)
Size of atlas image: (256, 256, 256)
```

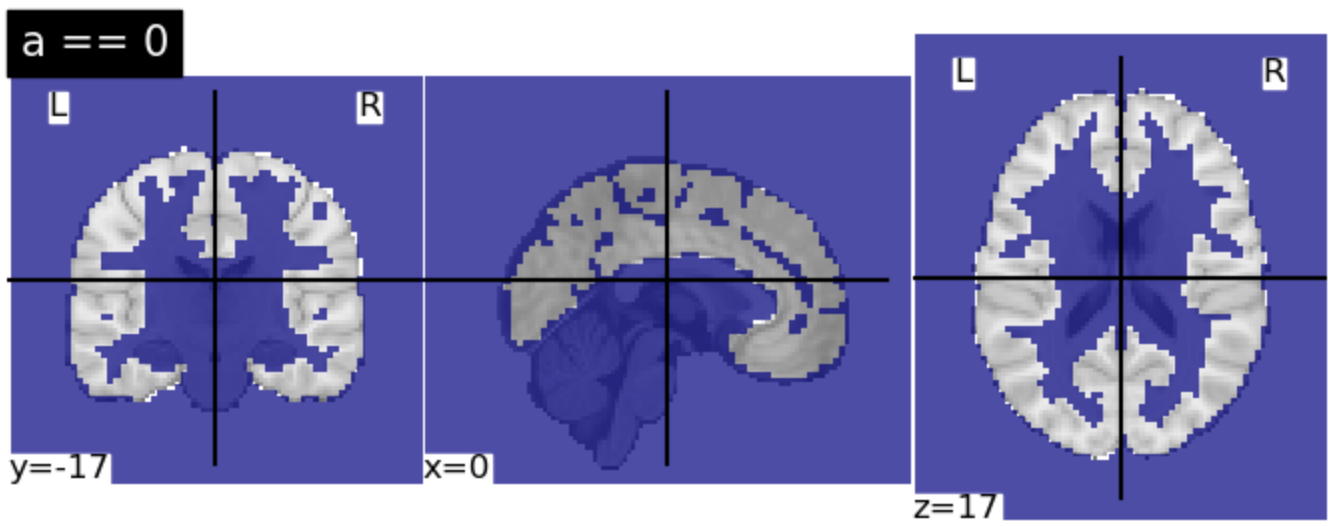
In [139…
```
# what the resampled atlas looks like overlayed on a slice of our NifTI file
# Note that we're pulling a random timepoint from the fMRI data
nplot.plot_roi(resampled_yeo, func_img.slicer[:, :, :, 54])
```

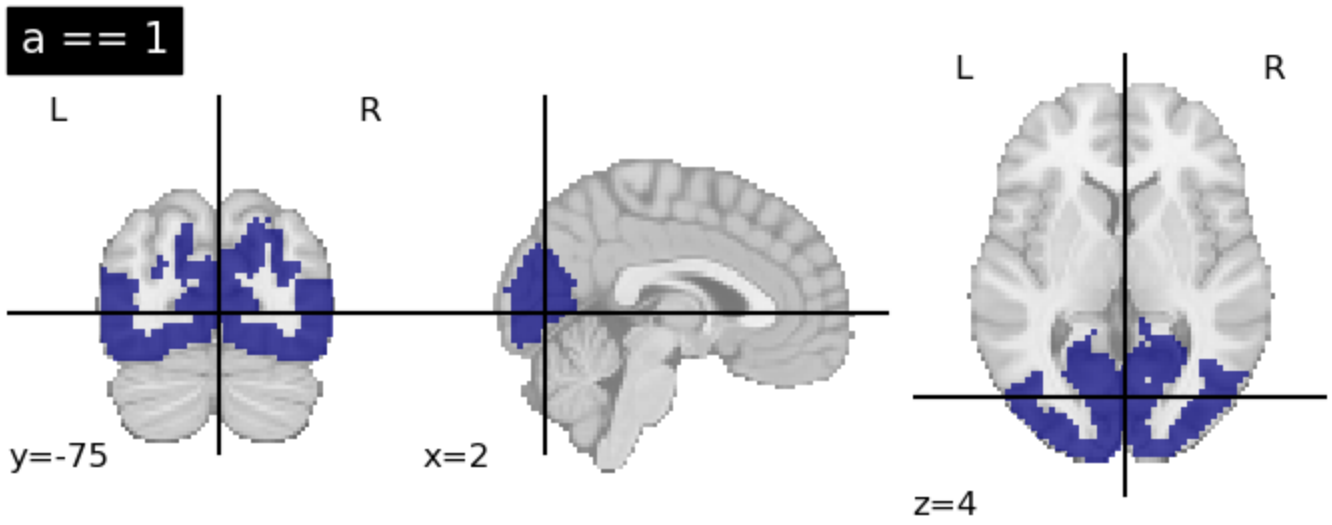
Out[139]:
```
<nilearn.plotting.displays._slicers.OrthoSlicer at 0x2c21f91e790>
```

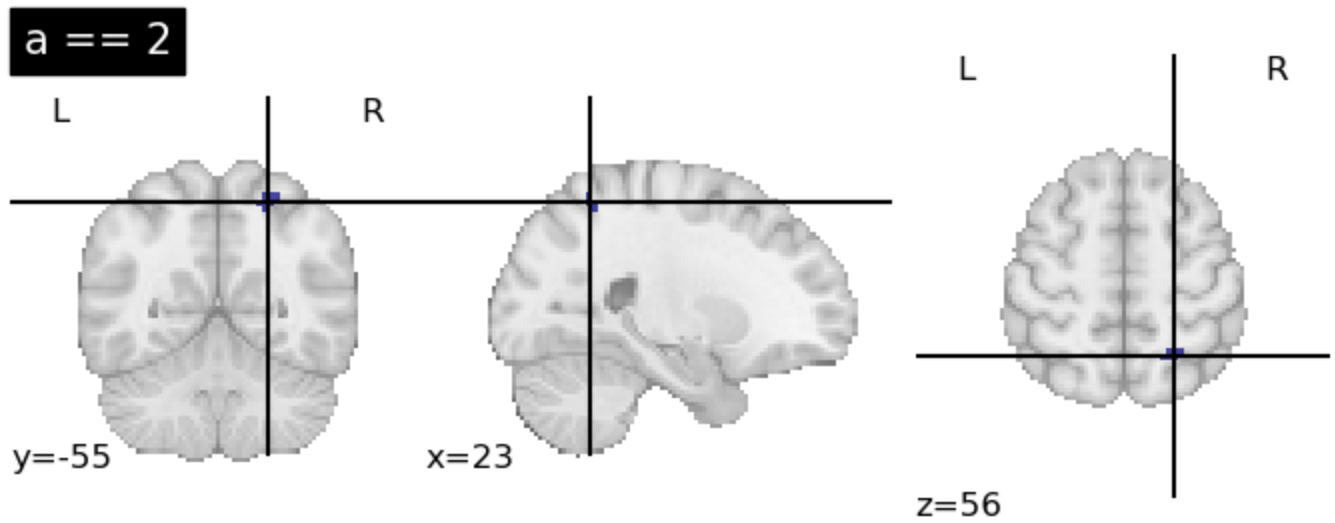


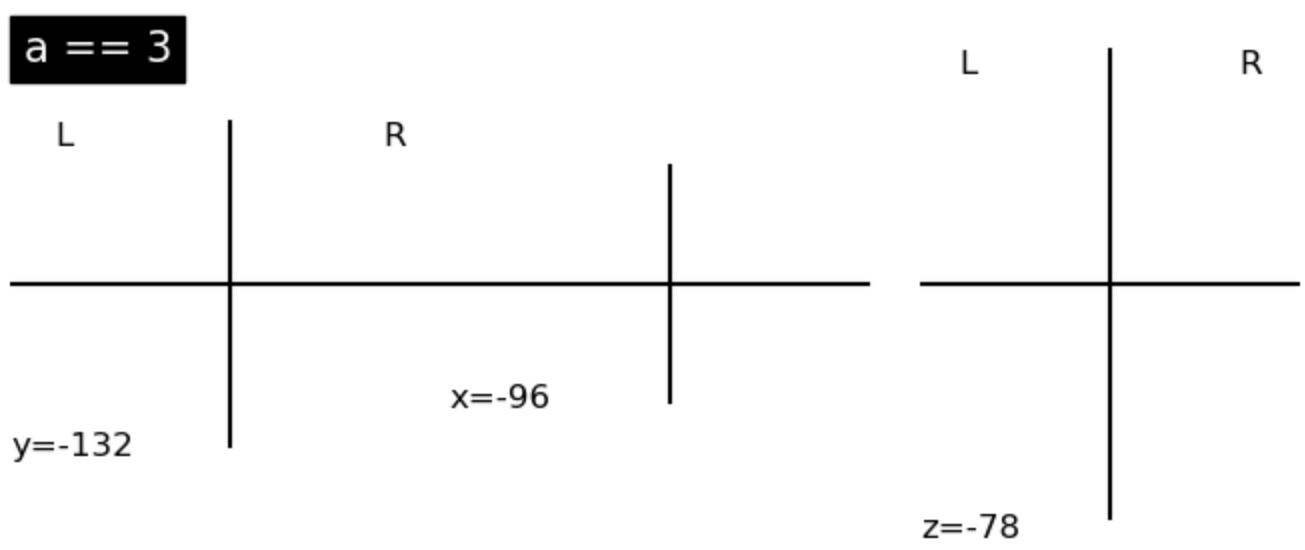
In [140…

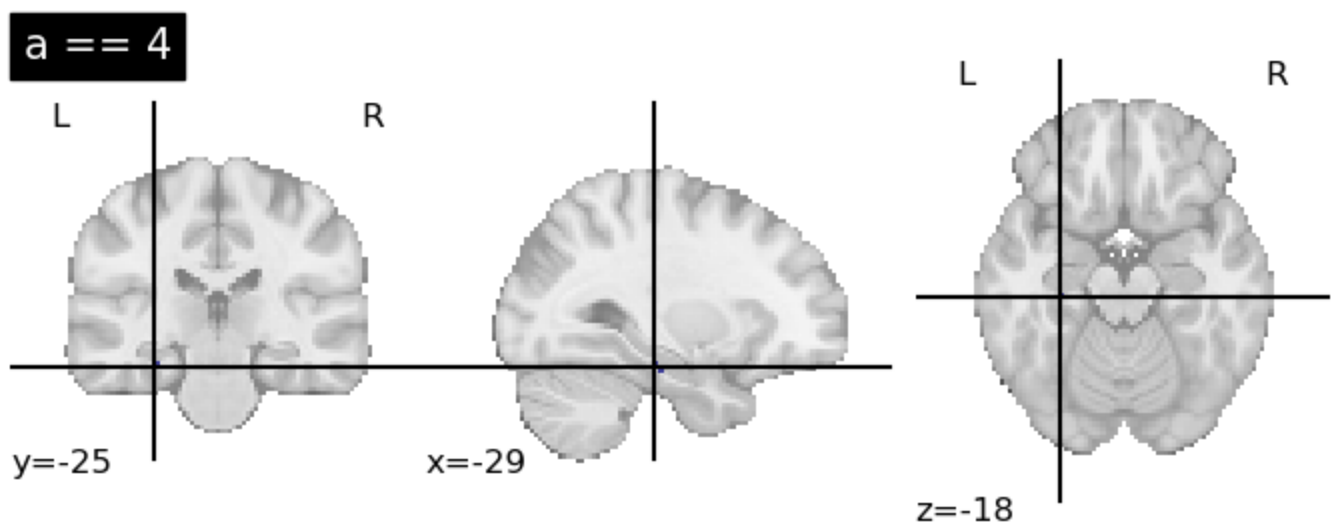
```
for i in range(50):
    a = "a == " + str(i)
    # print("ROI: " + a)
    # Make a mask for ROI
    roi_mask = nimg.math_img(a, a=resampled_yeo)
    # Visualize ROI
    nplot.plot_roi(roi_mask, title = a)
```



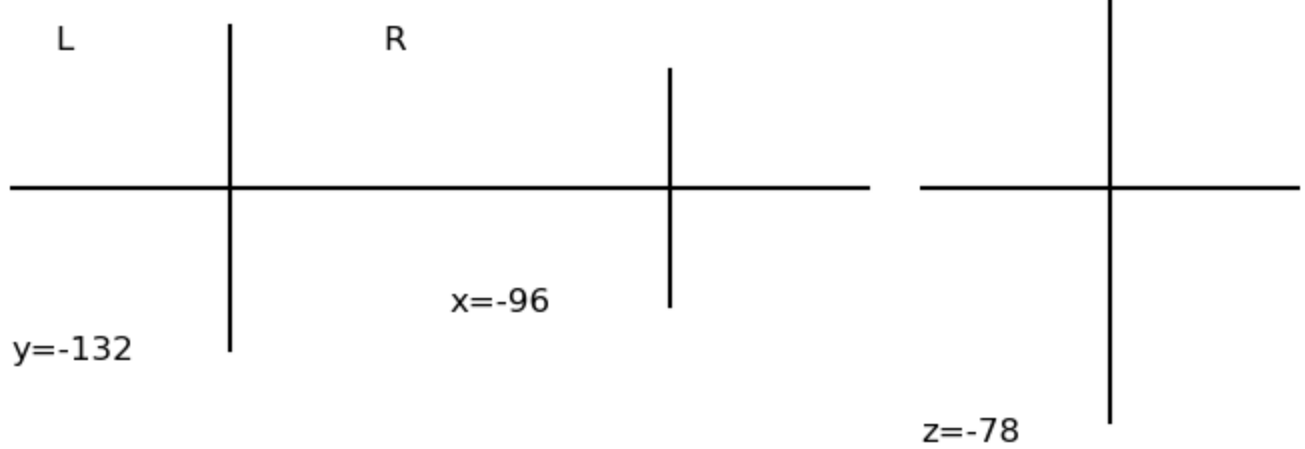



a == 2
L
R
y=-55
x=23
L
R
z=56

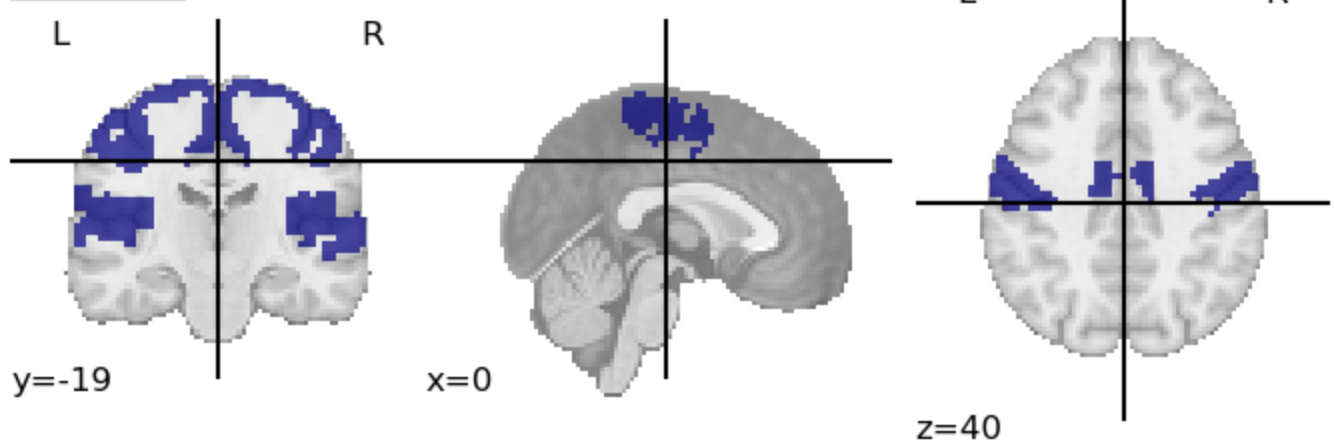

a == 3
L
R
y=-132
x=-96
L
R
z=-78


a == 4
L
R
y=-25
x=-29
L
R
z=-18

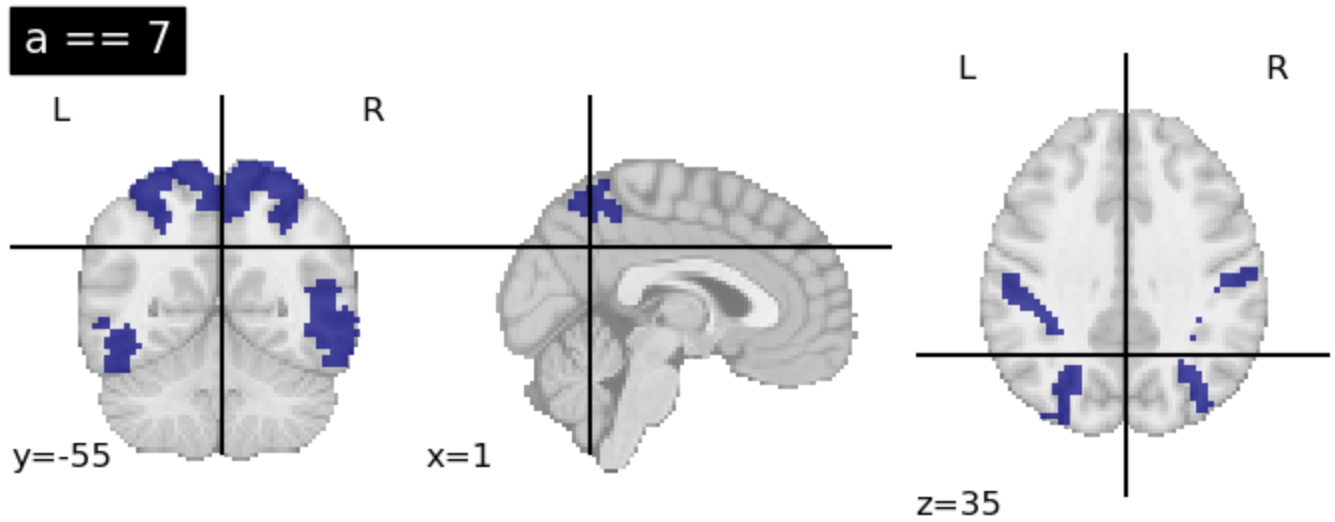
a == 5



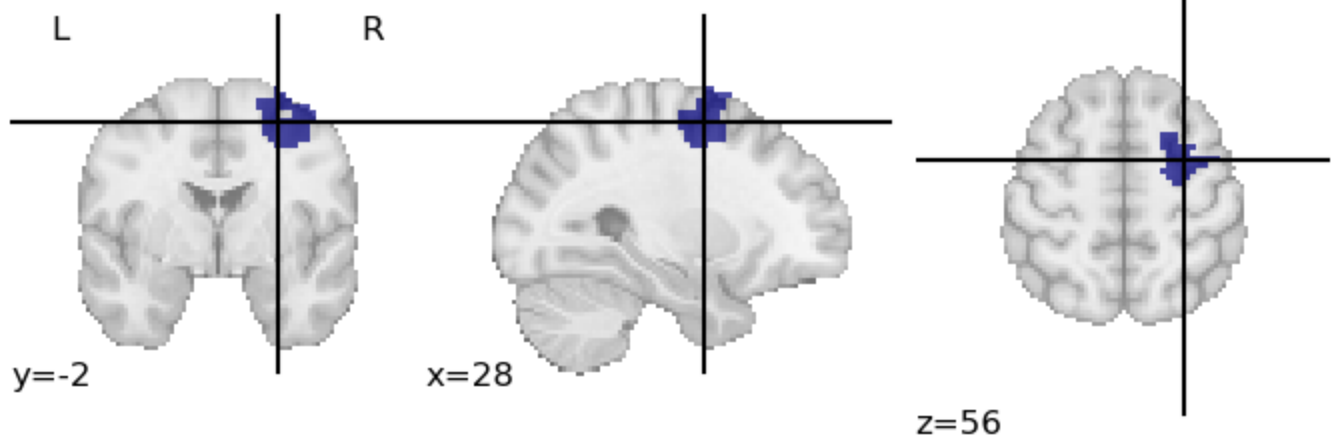
a == 6



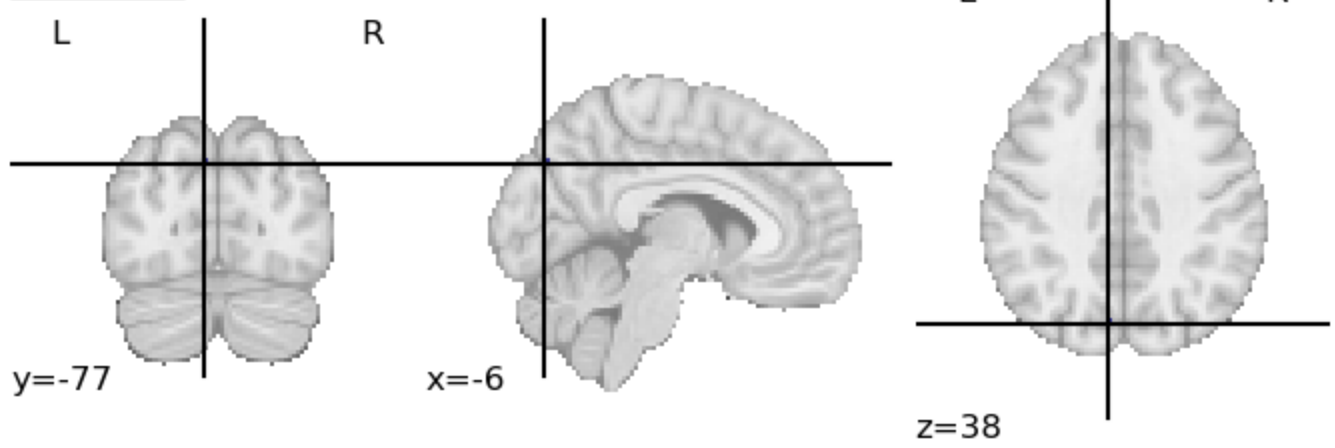
a == 7



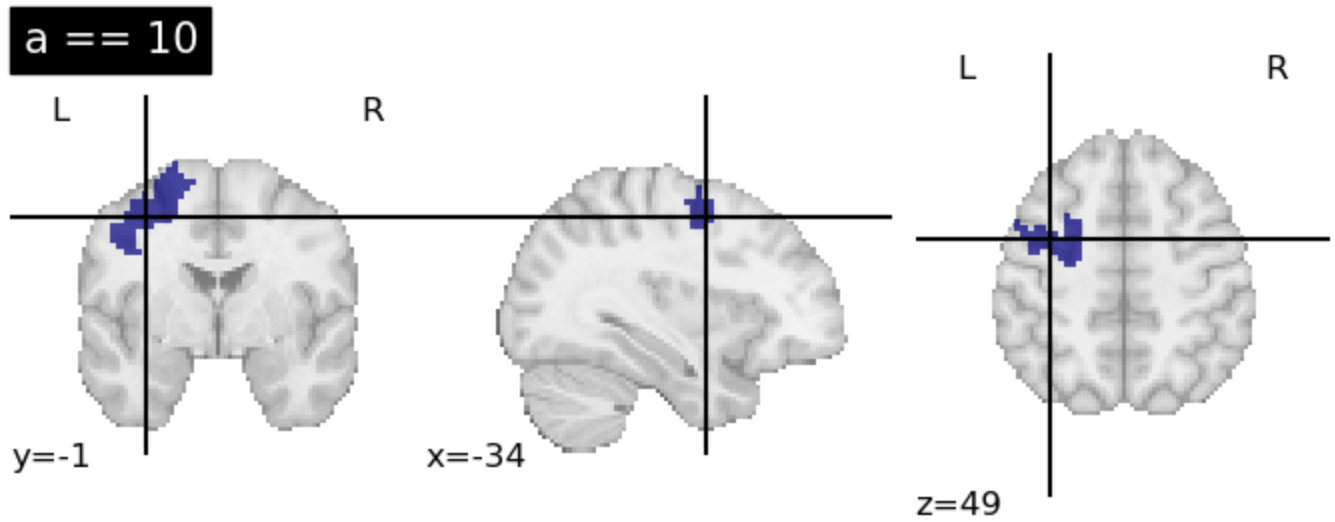
a == 8

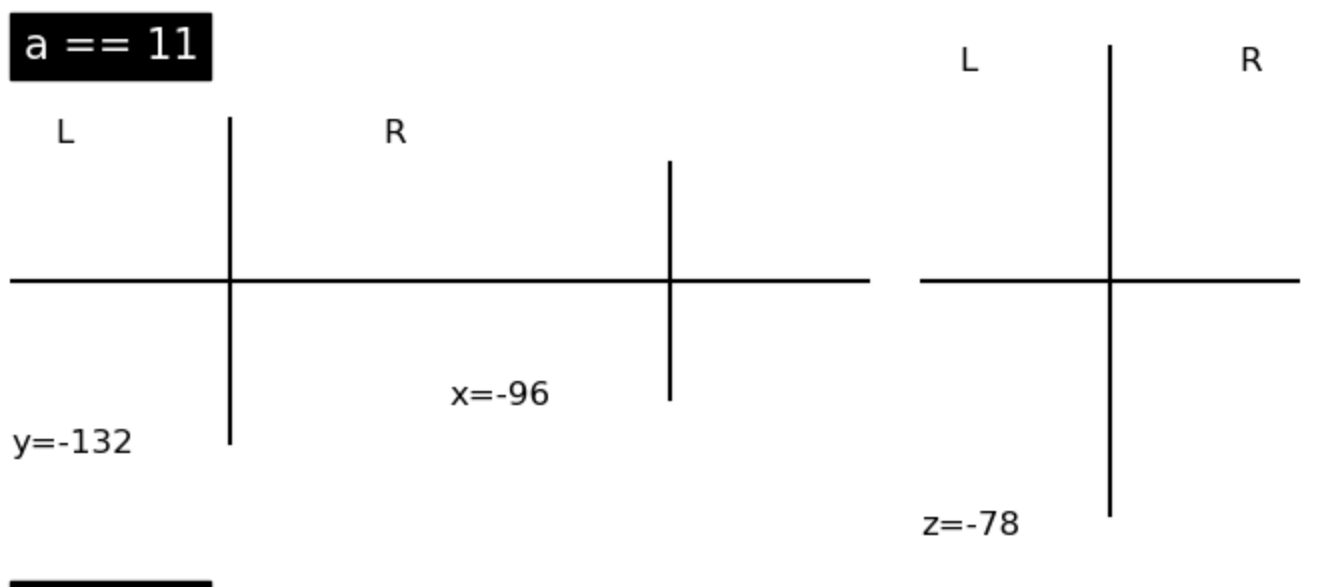


a == 9

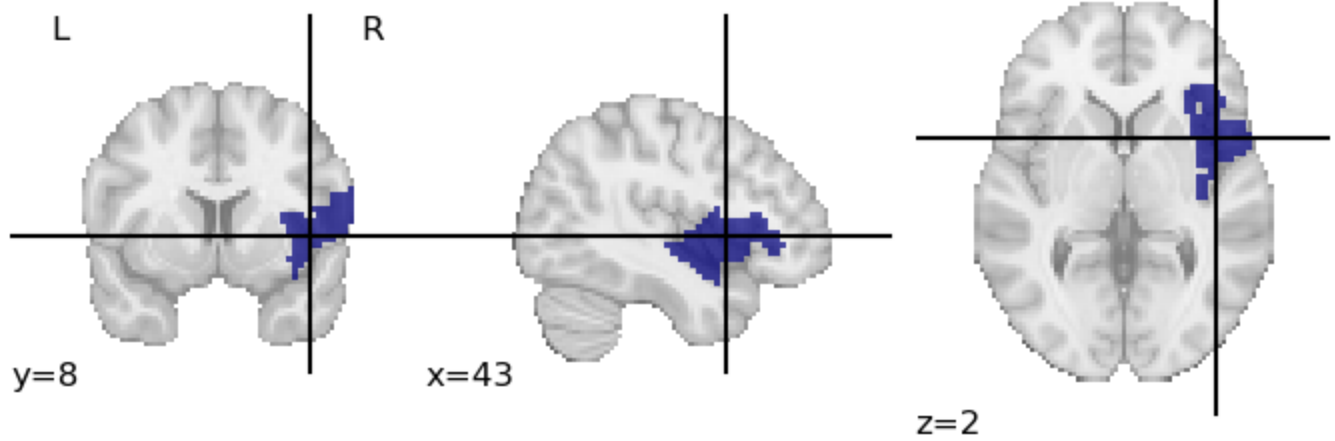


a == 10

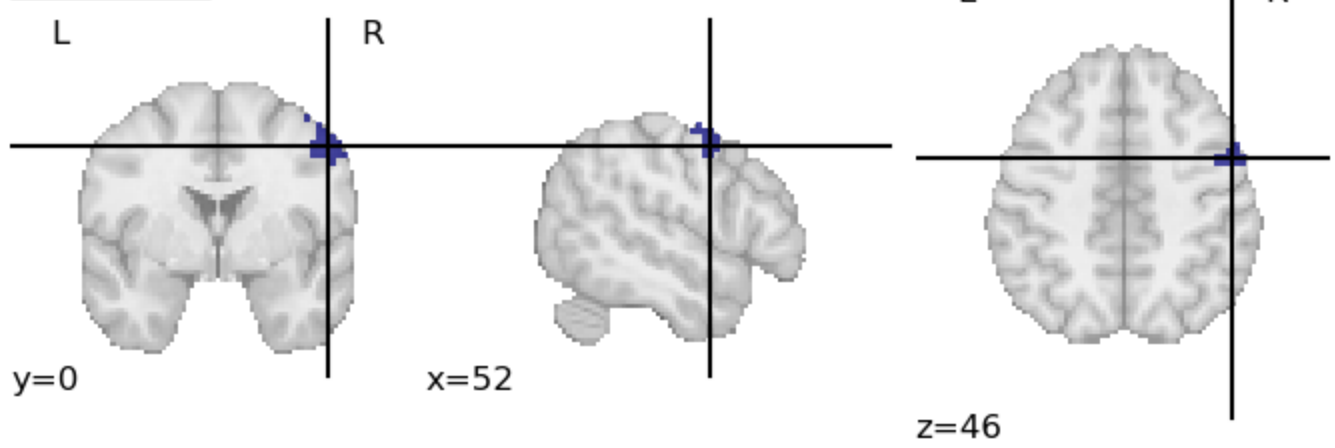



a == 11
L
R
y=-132
x=-96
L
R
z=-78
a == 12
L
R
y=-28
x=-34
L
R
z=-19
a == 13
L
R
y=-32
x=60
L
R
z=23

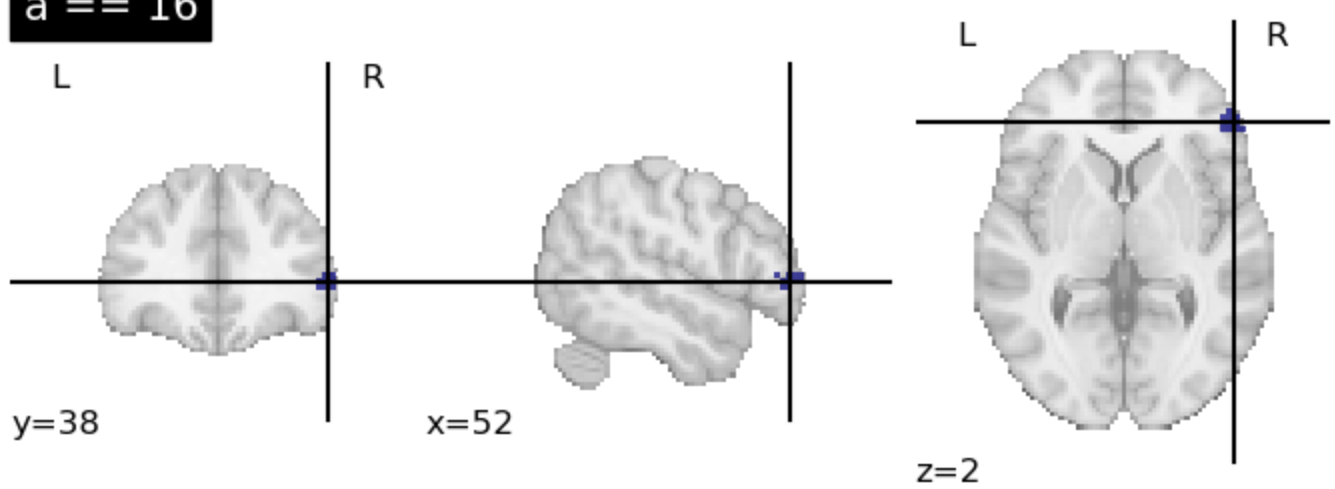
a == 14

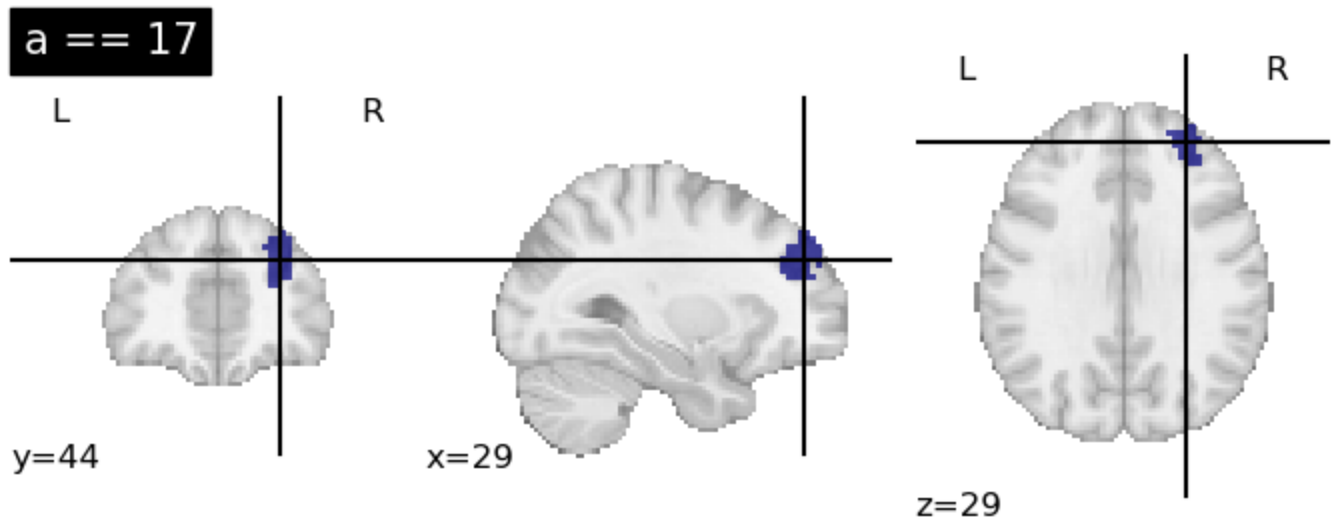


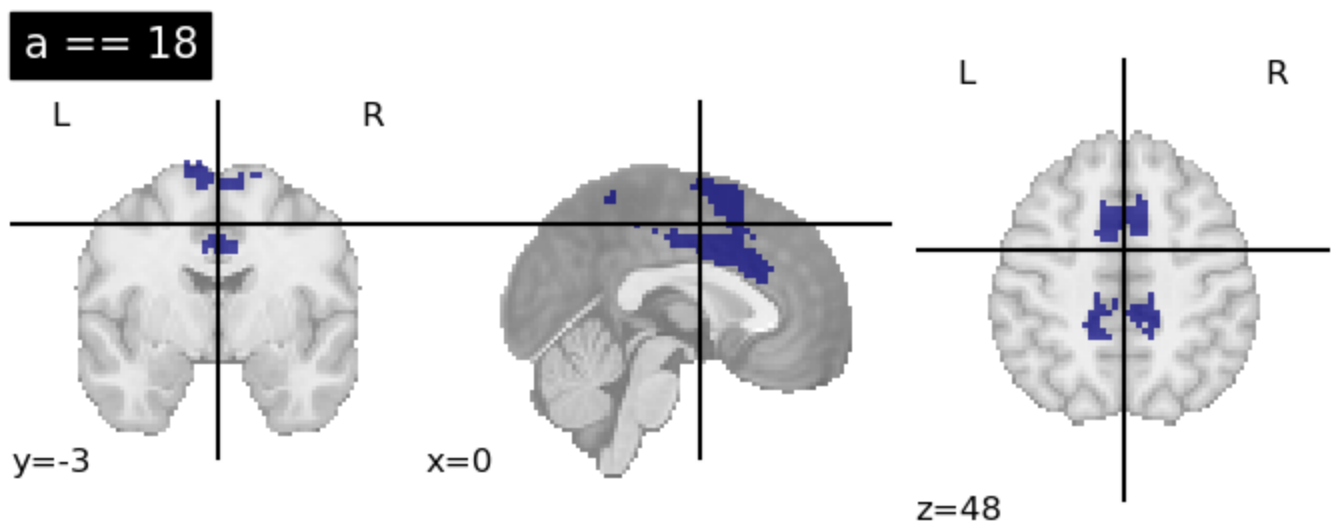
a == 15

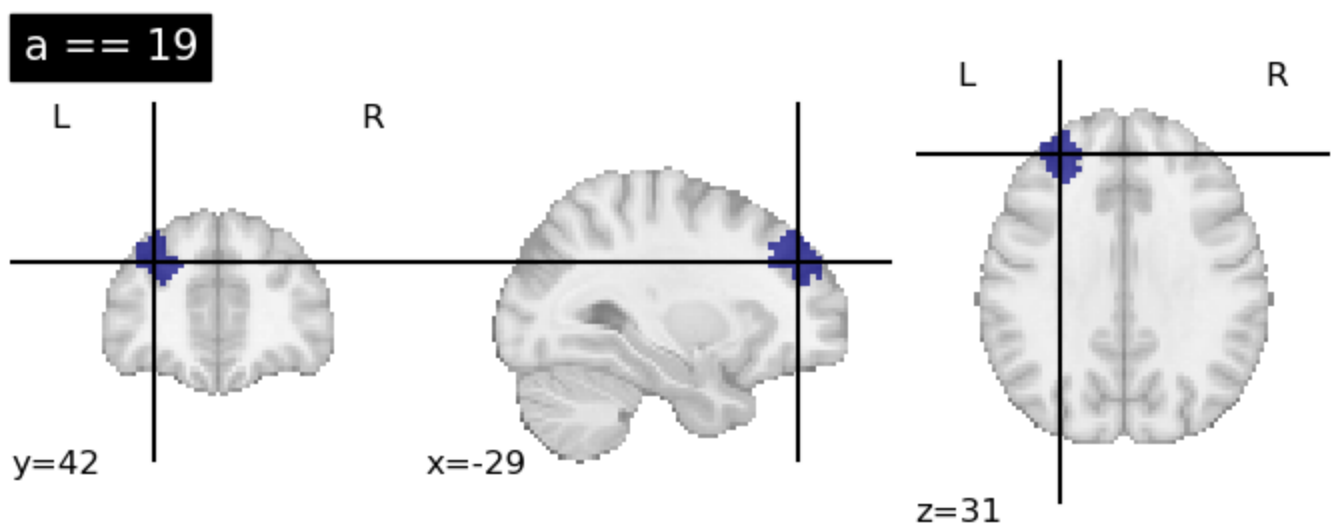


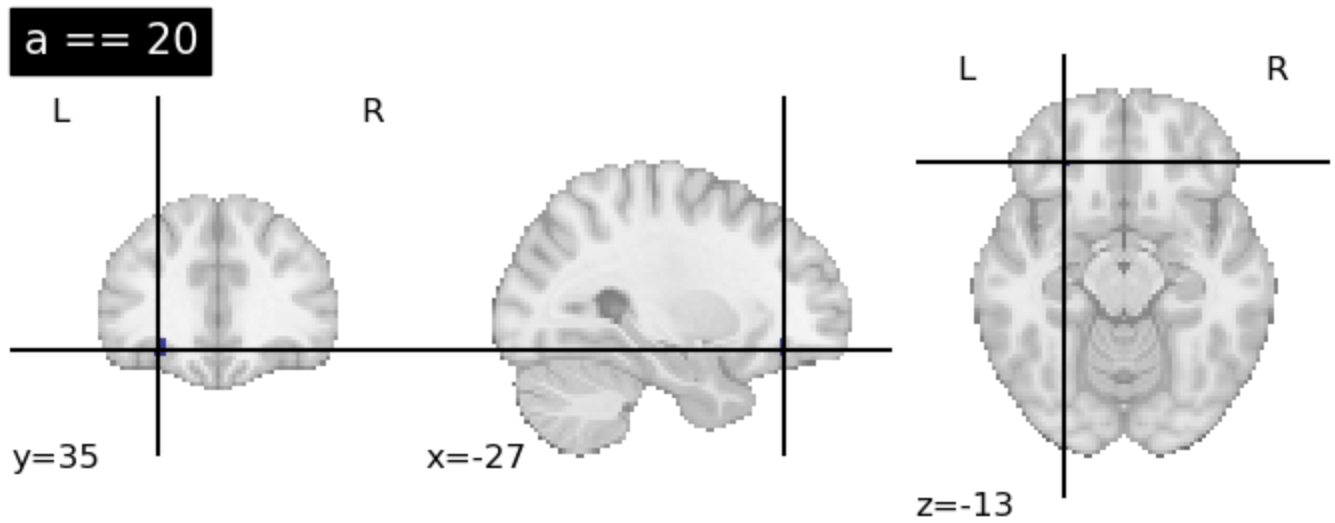
a == 16

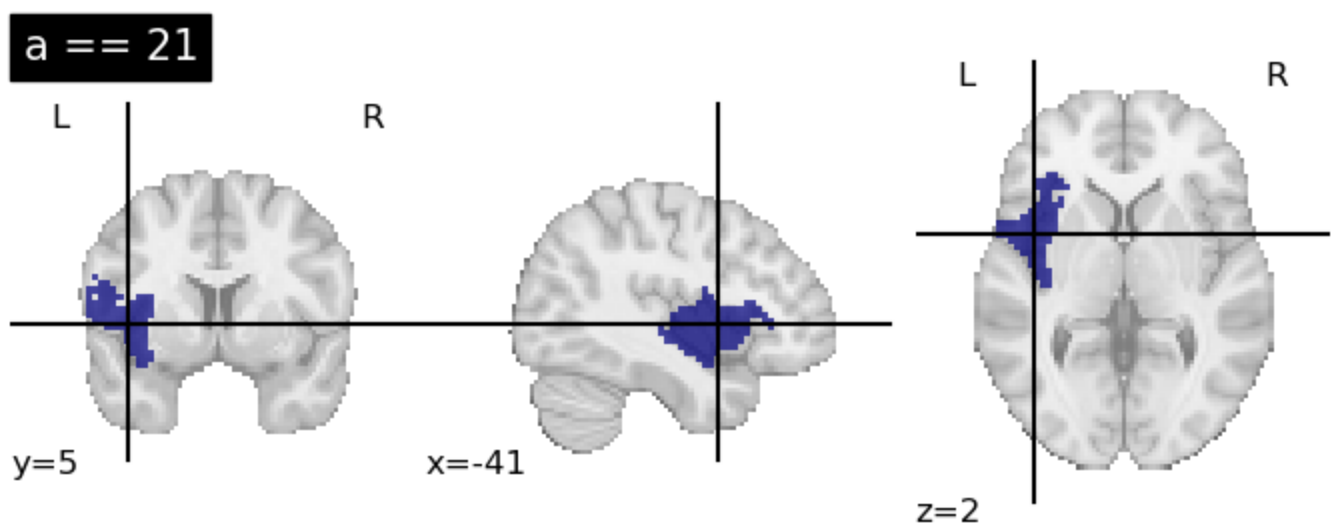


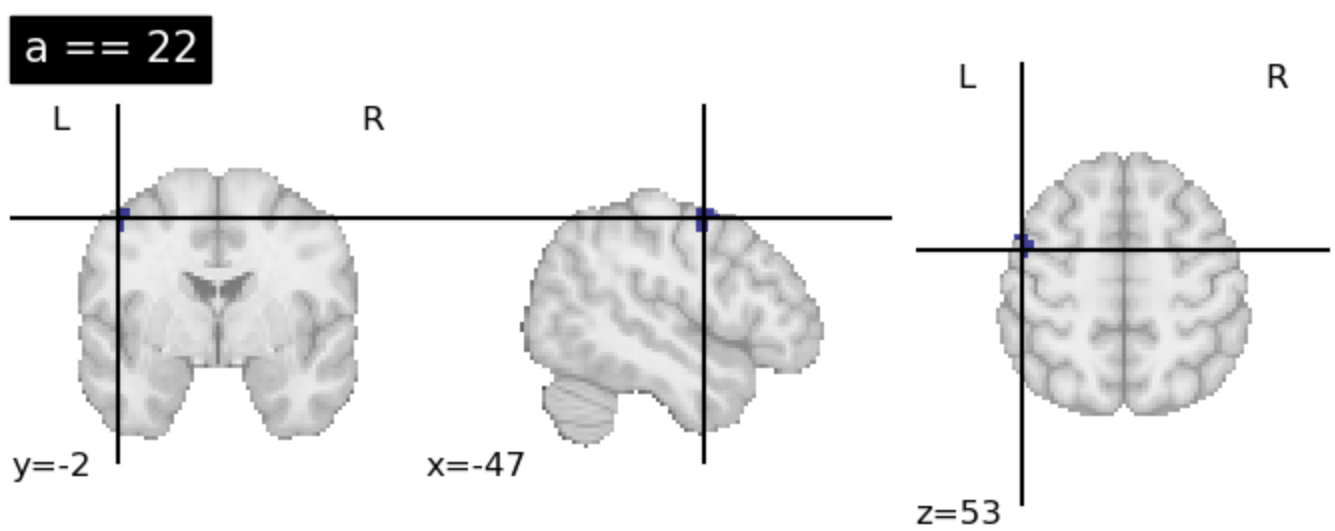

a == 17
L
R
y=44
x=29
L
R
z=29

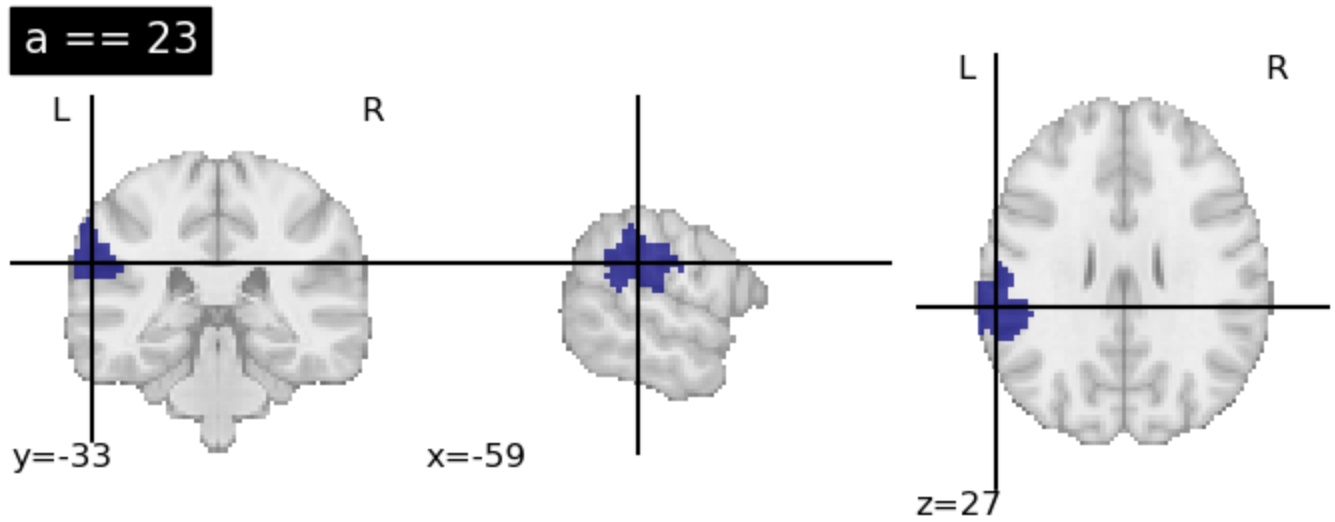

a == 18
L
R
y=-3
x=0
L
R
z=48

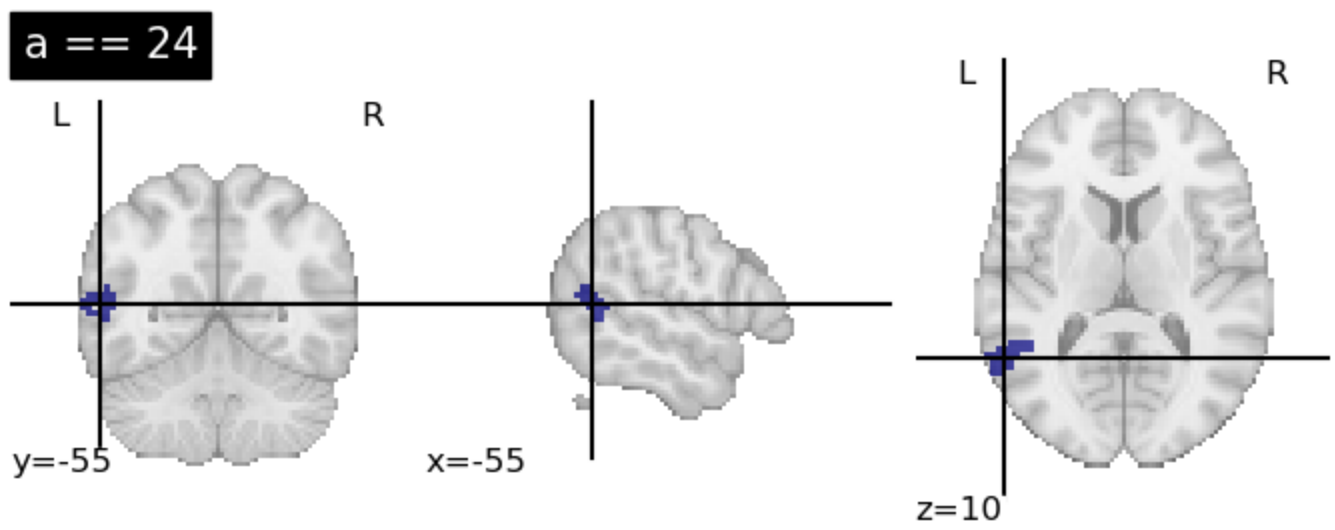

a == 19
L
R
y=42
x=-29
L
R
z=31

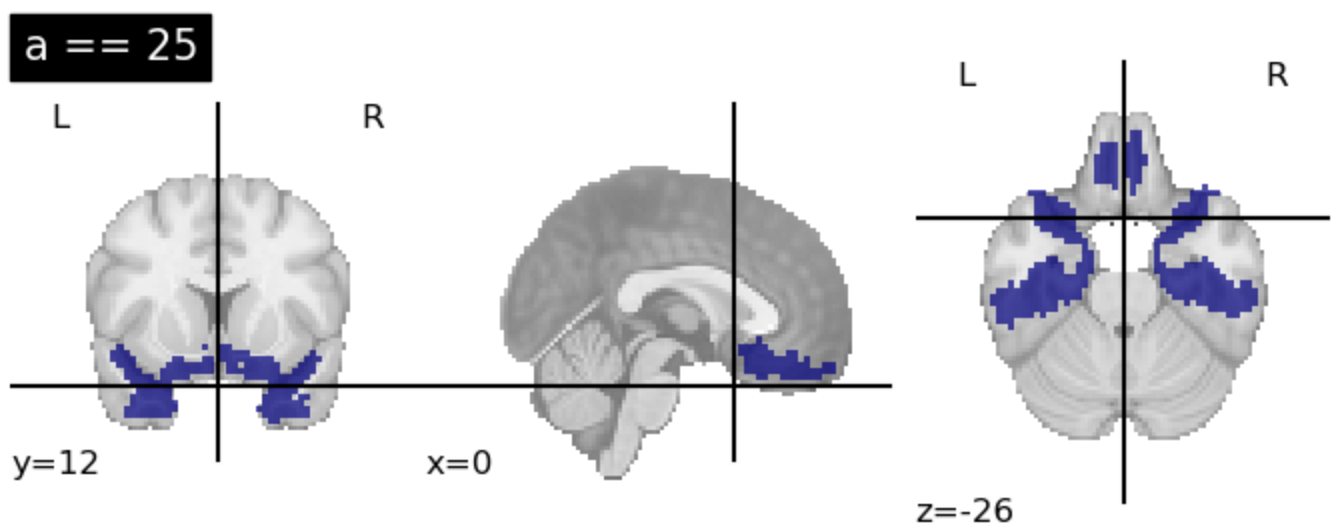

a == 20
L
R
y=35
x=-27
L
R
z=-13

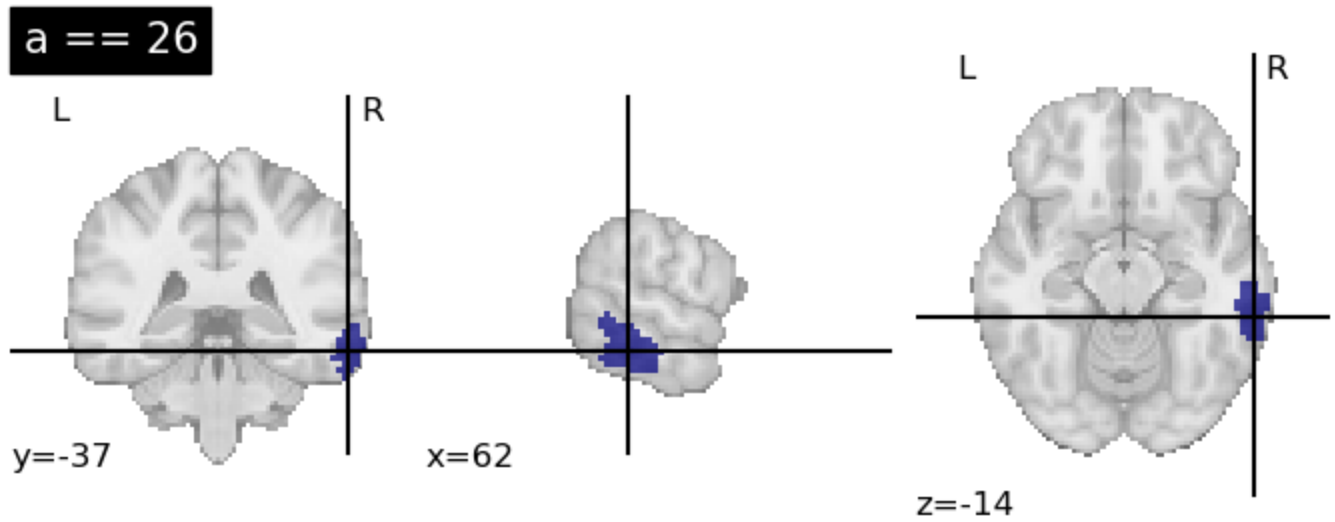

a == 21
L
R
y=5
x=-41
L
R
z=2

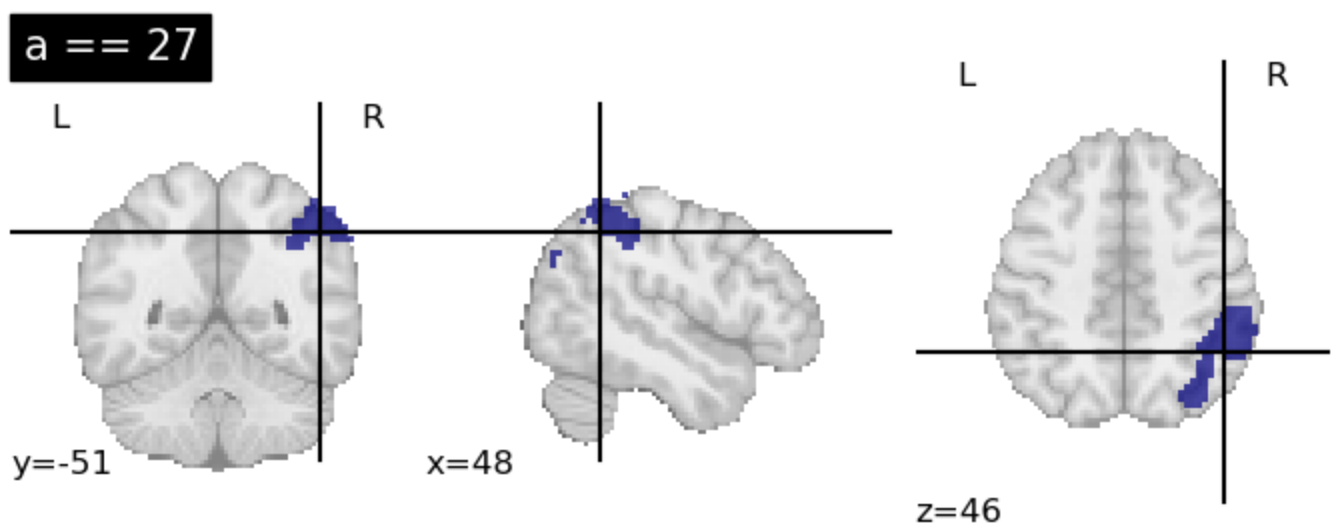

a == 22
L
R
y=-2
x=-47
L
R
z=53

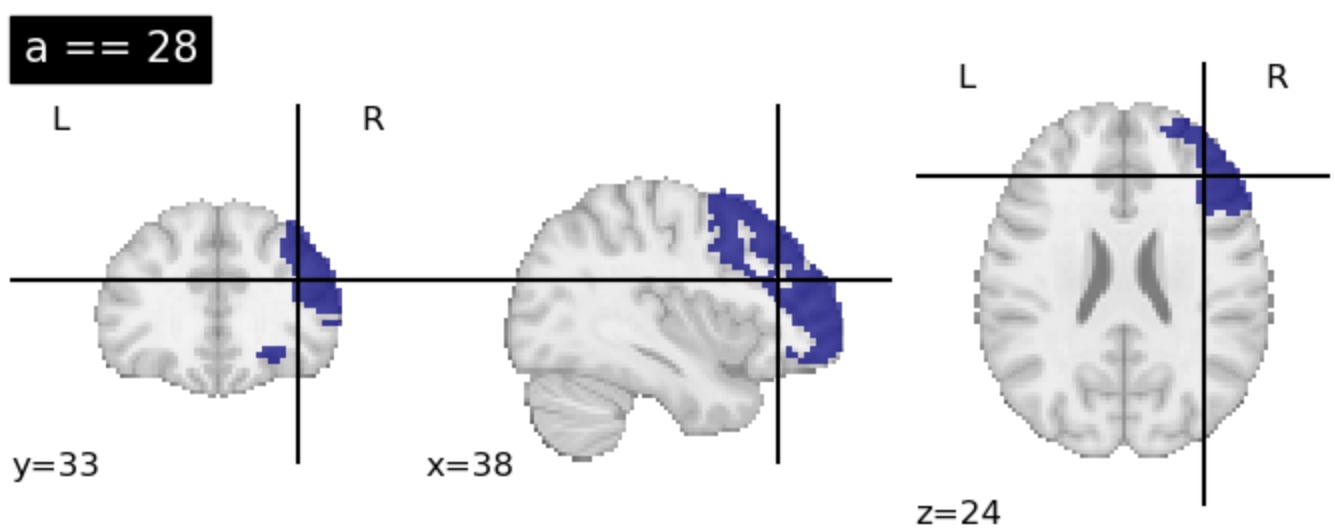

a == 23
L
R
y=-33
x=-59
L
R
z=27

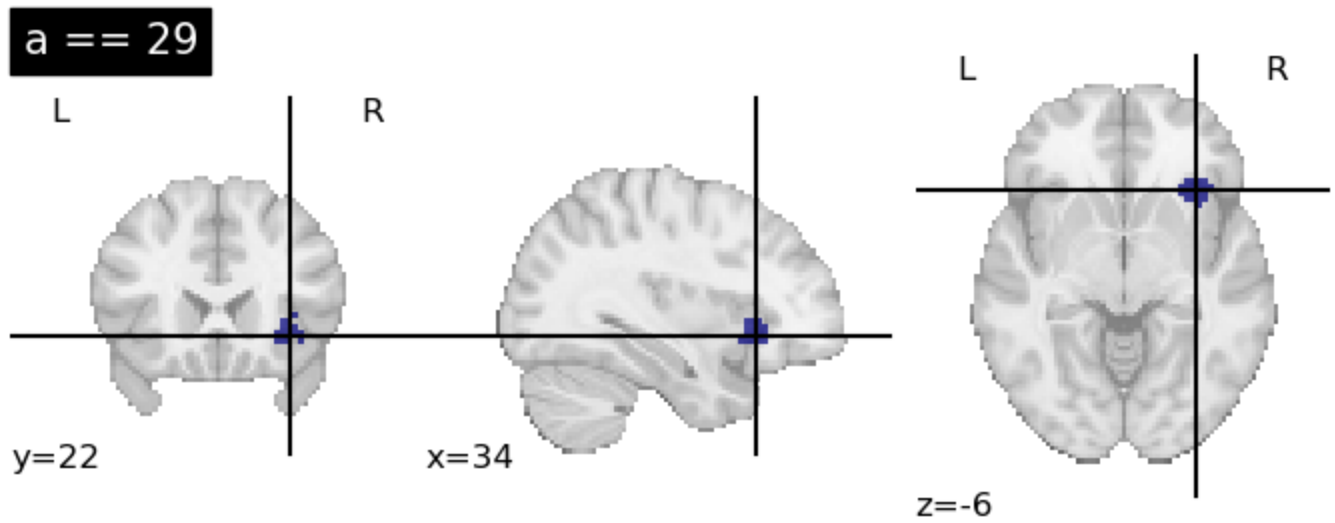

a == 24
L
R
y=-55
x=-55
L
R
z=10

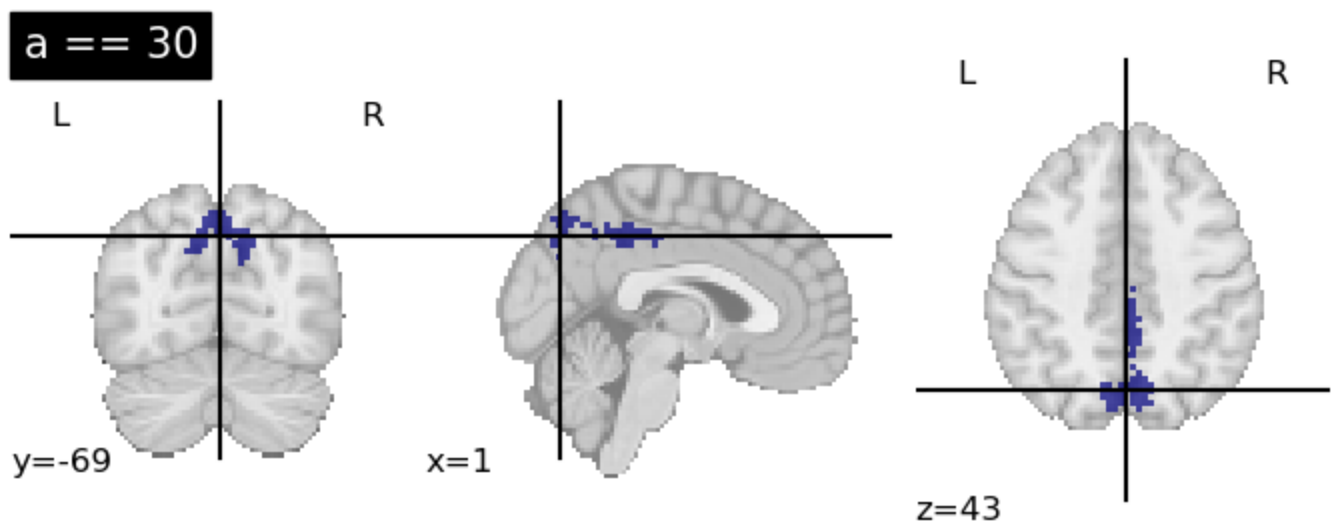

a == 25
L
R
y=12
x=0
L
R
z=-26

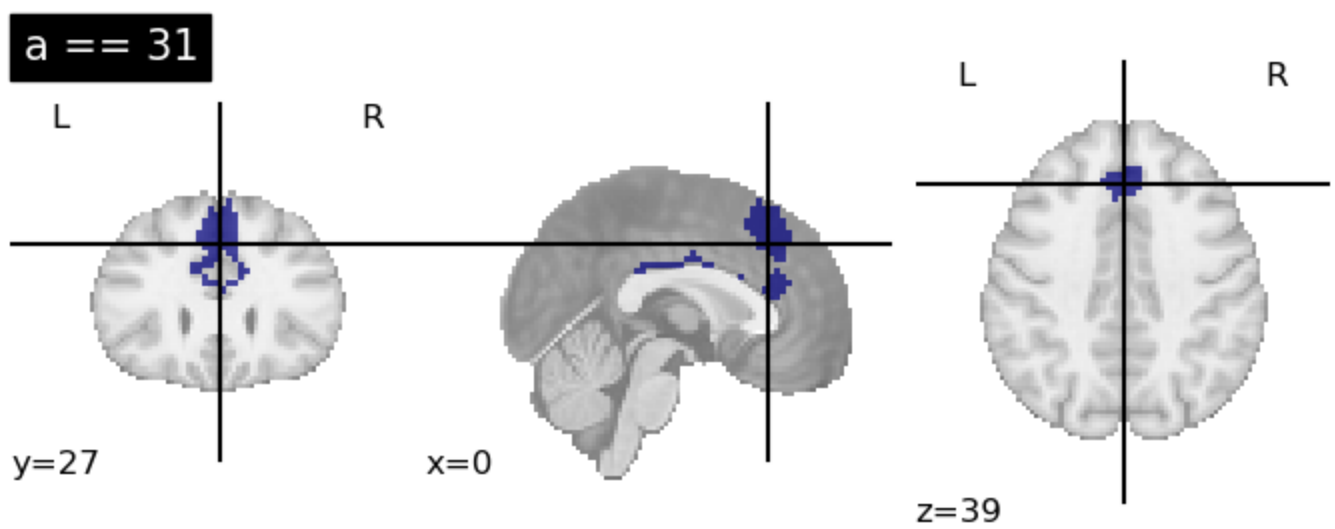

a == 26
L
R
y=-37
x=62
L
R
z=-14


a == 27
L
R
y=-51
x=48
L
R
z=46

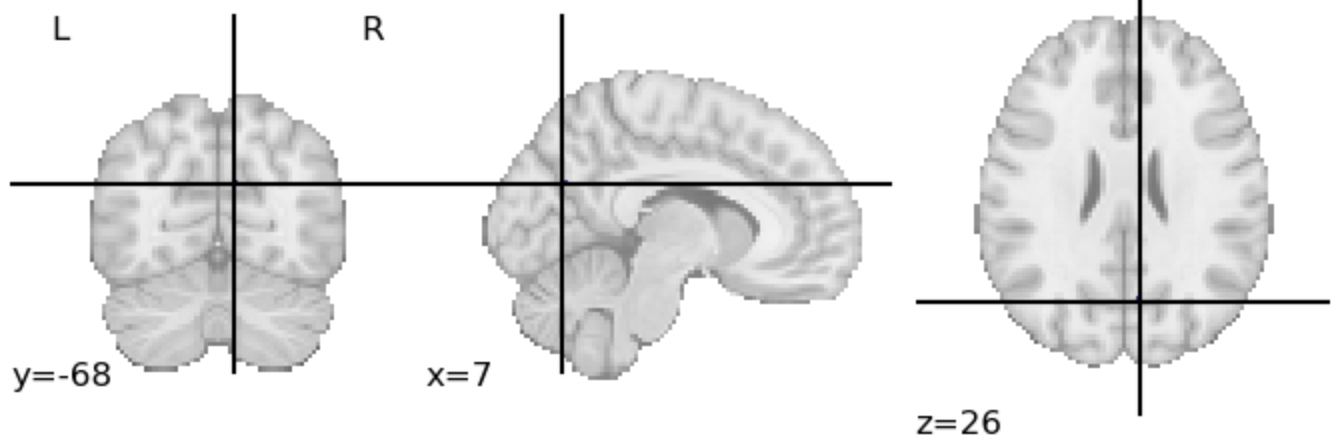

a == 28
L
R
y=33
x=38
L
R
z=24


a == 29
L
R
y=22
x=34
L
R
z=-6

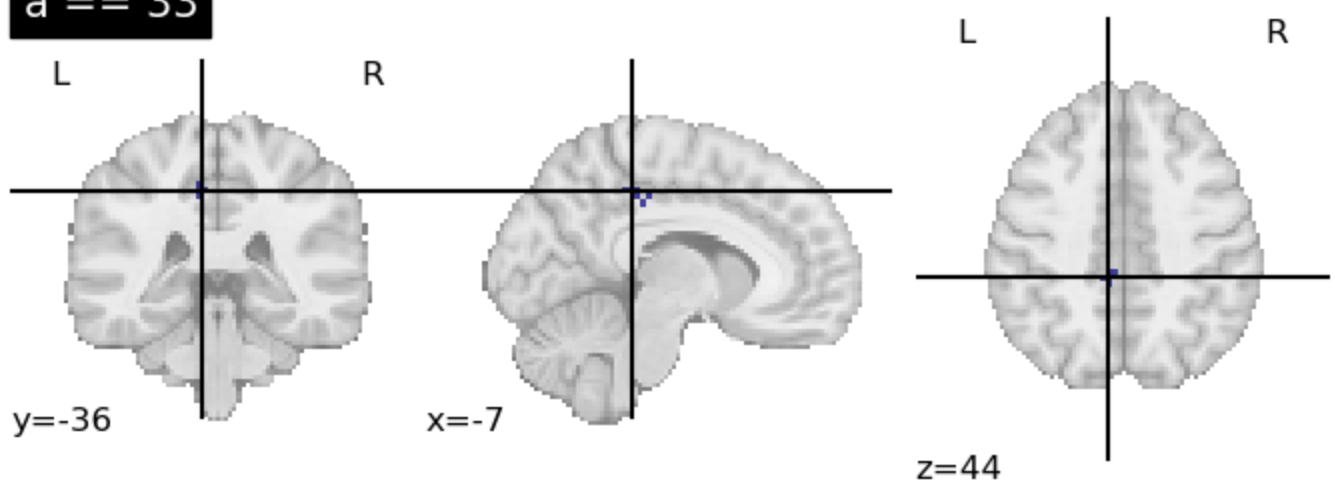

a == 30
L
R
y=-69
x=1
L
R
z=43


a == 31
L
R
y=27
x=0
L
R
z=39

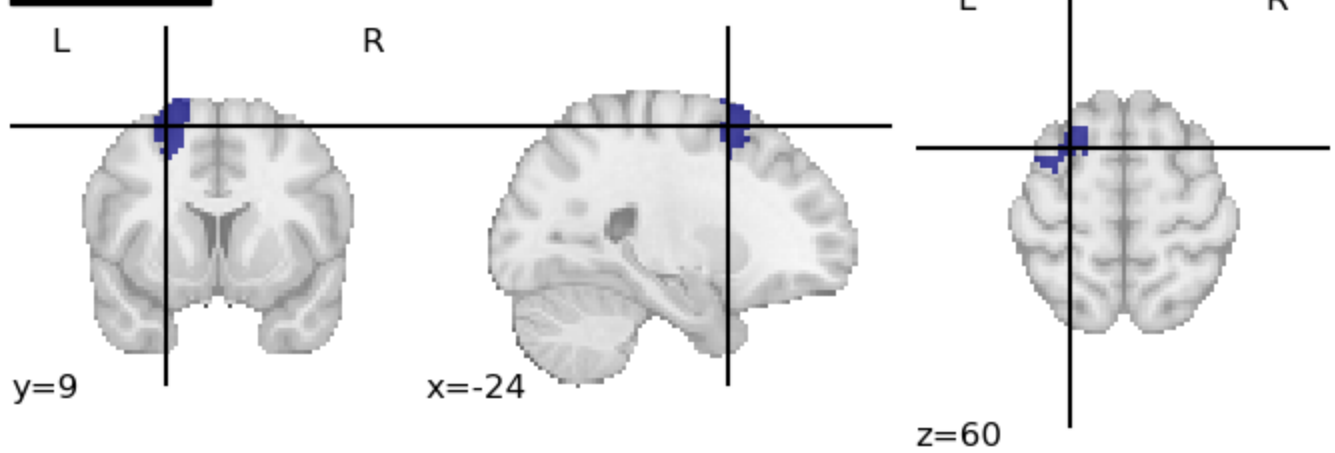
a == 32

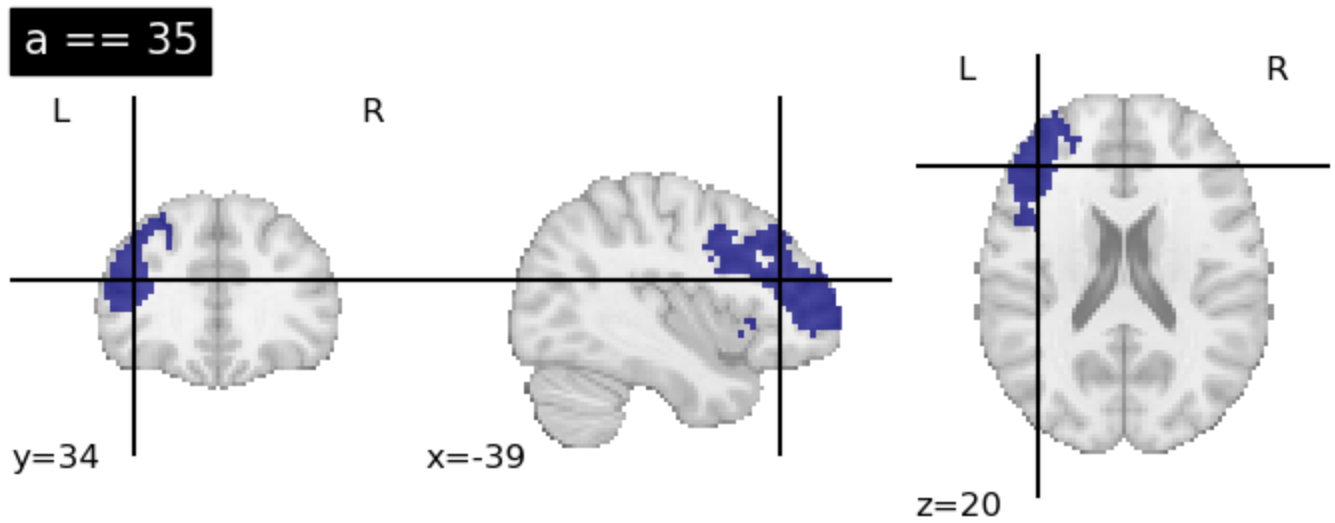


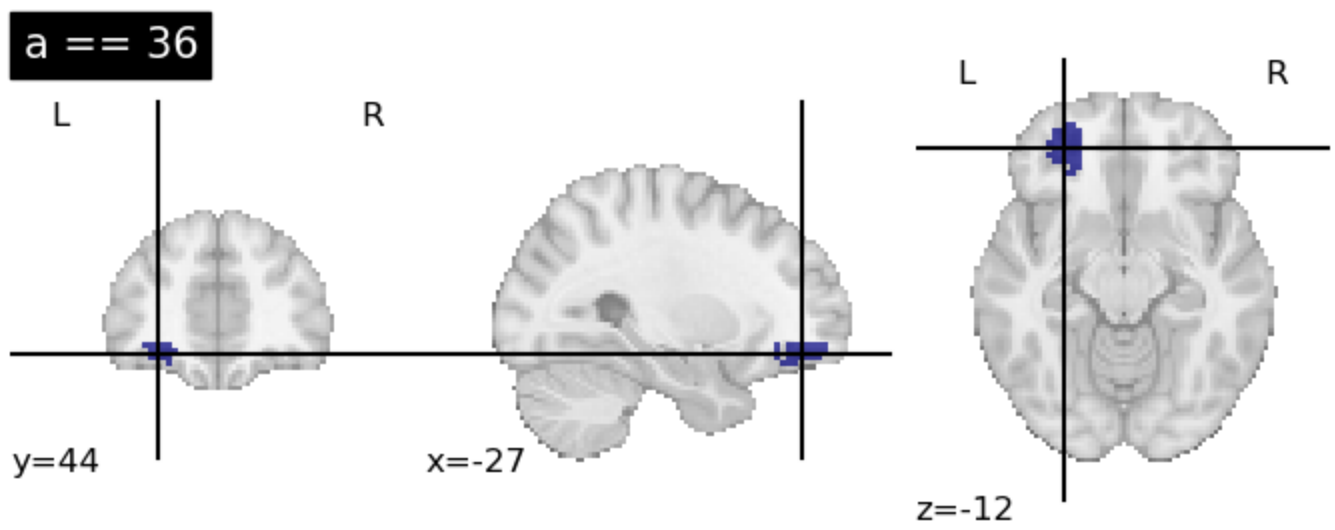
a == 33

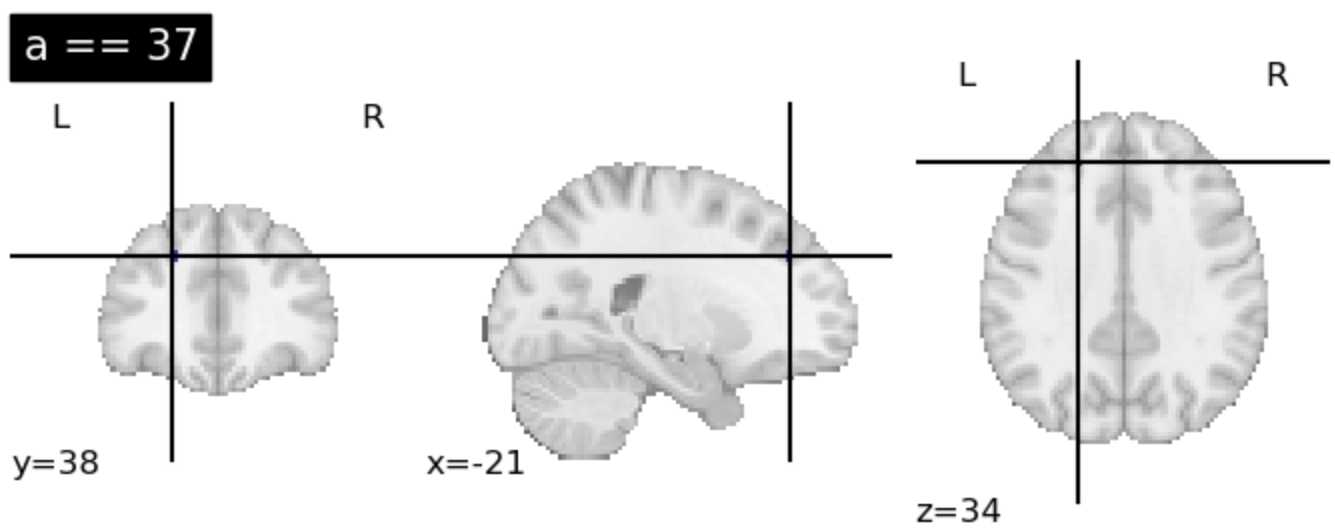


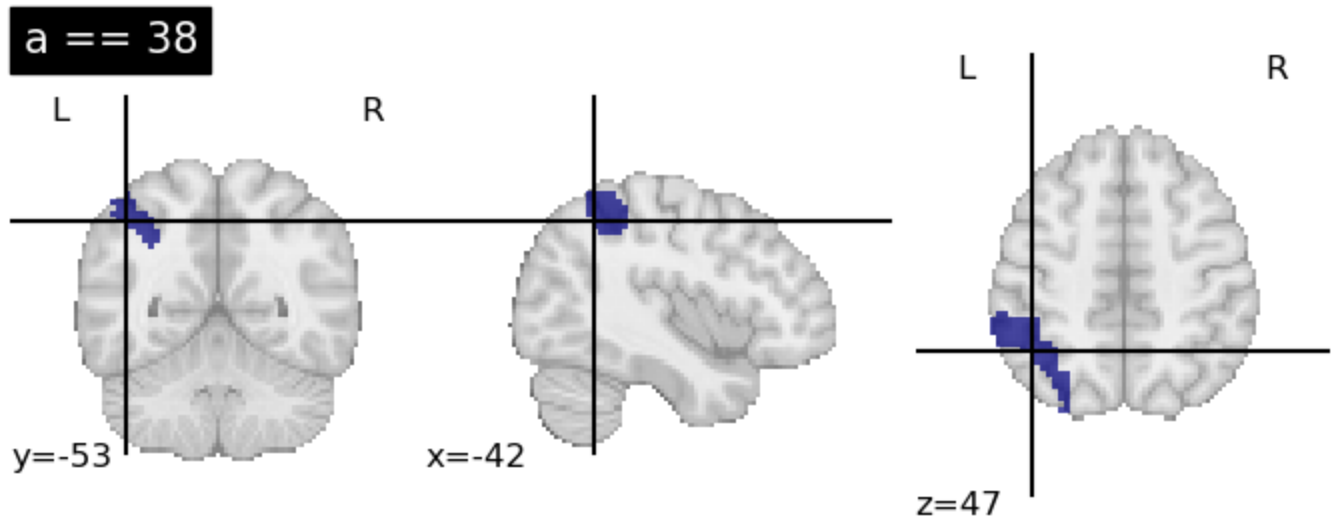
a == 34

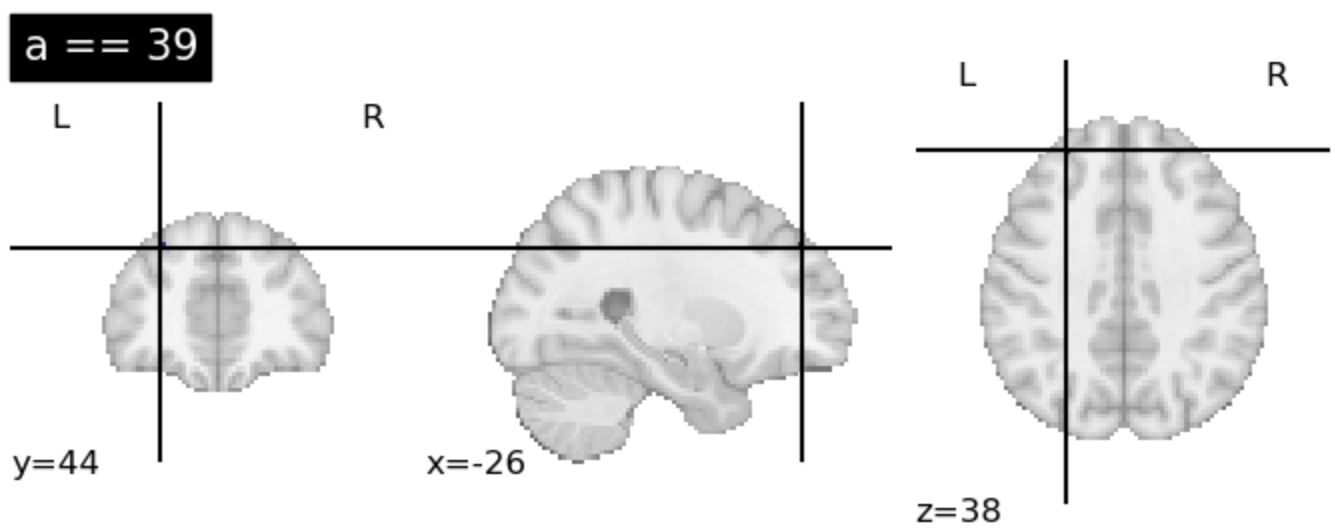


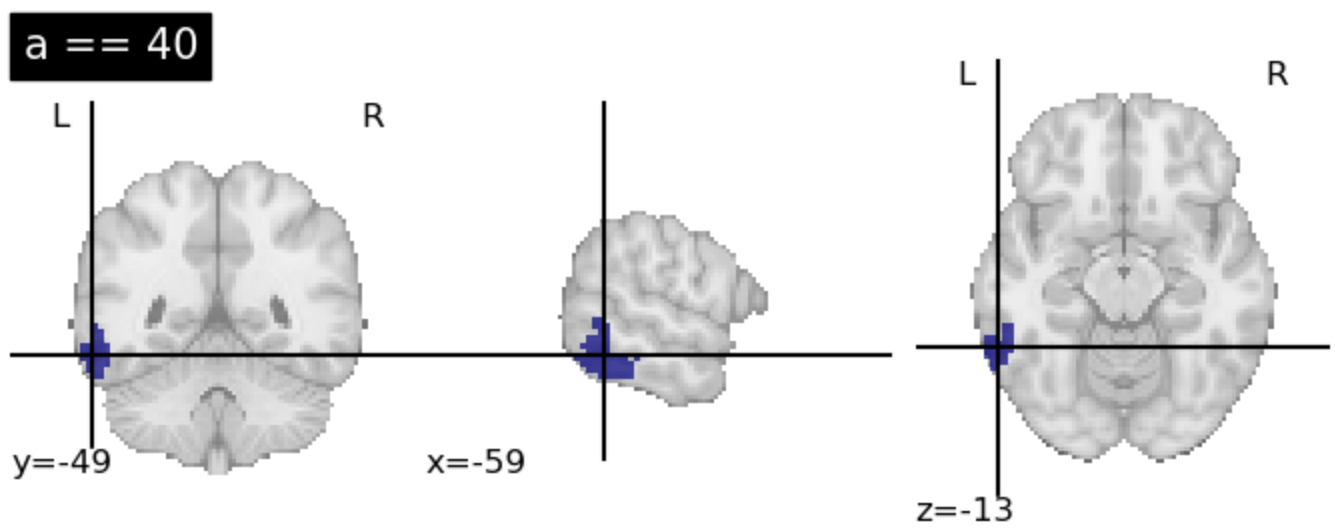

a == 35
L
R
y=34
x=-39
L
R
z=20

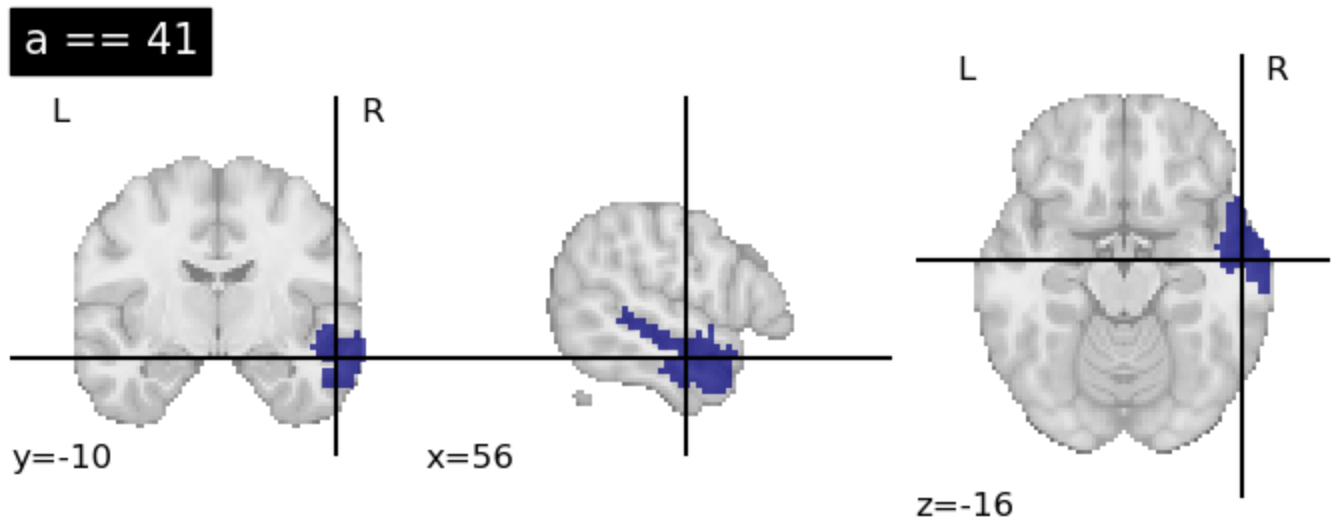

a == 36
L
R
y=44
x=-27
L
R
z=-12

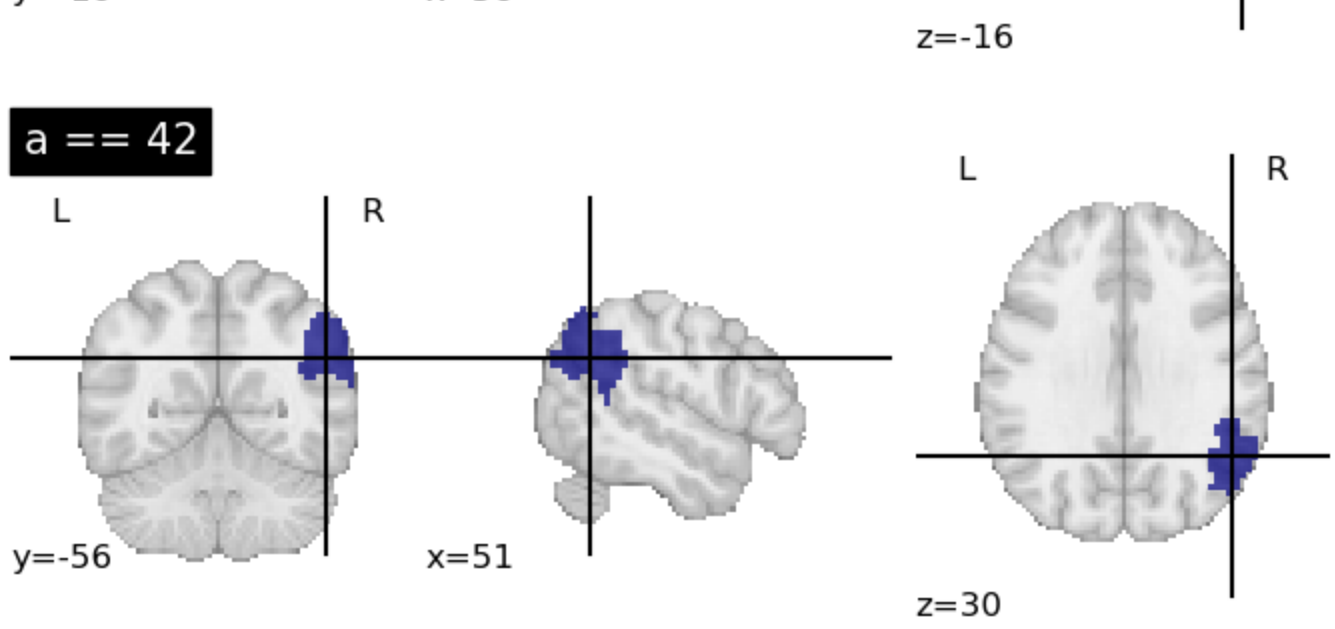

a == 37
L
R
y=38
x=-21
L
R
z=34

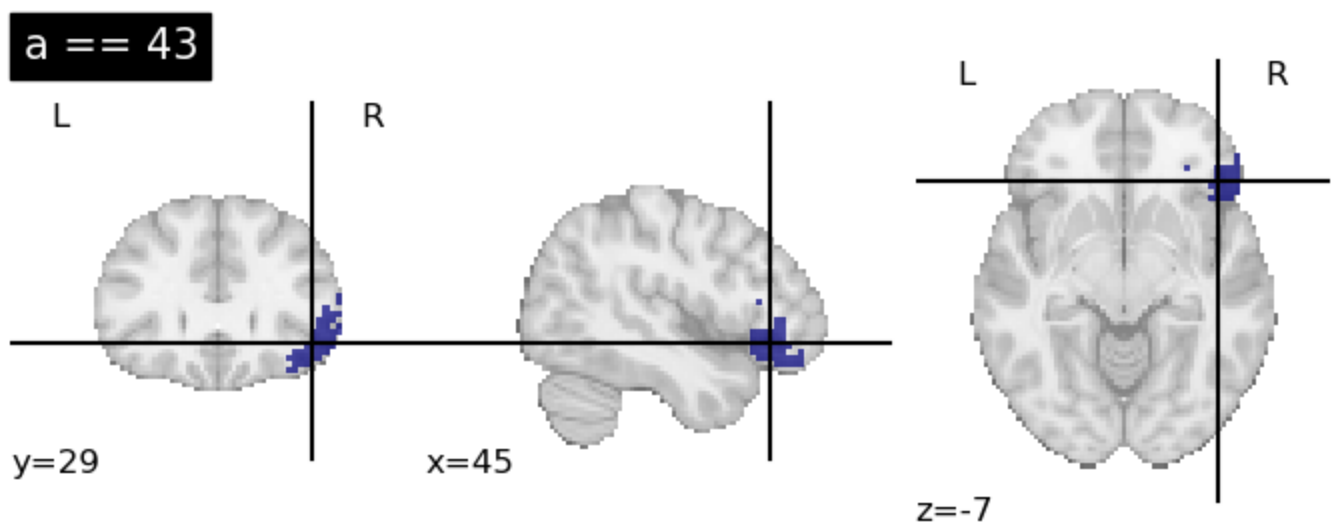

a == 38
L
R
y=-53
x=-42
L
R
z=47

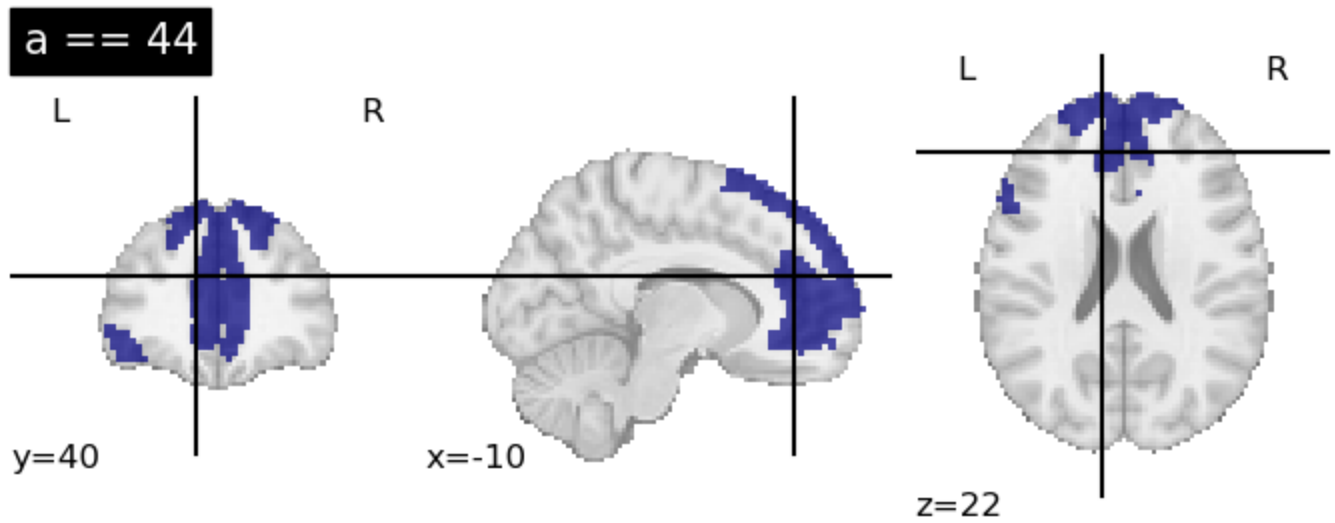

a == 39
L
R
y=44
x=-26
L
R
z=38

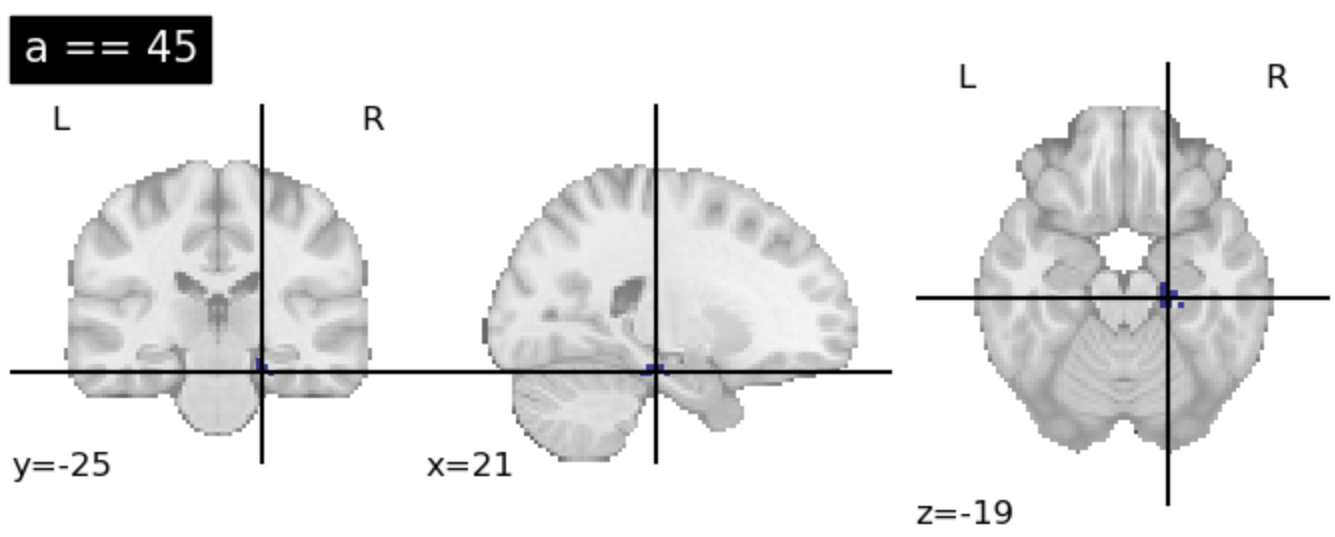

a == 40
L
R
y=-49
x=-59
L
R
z=-13

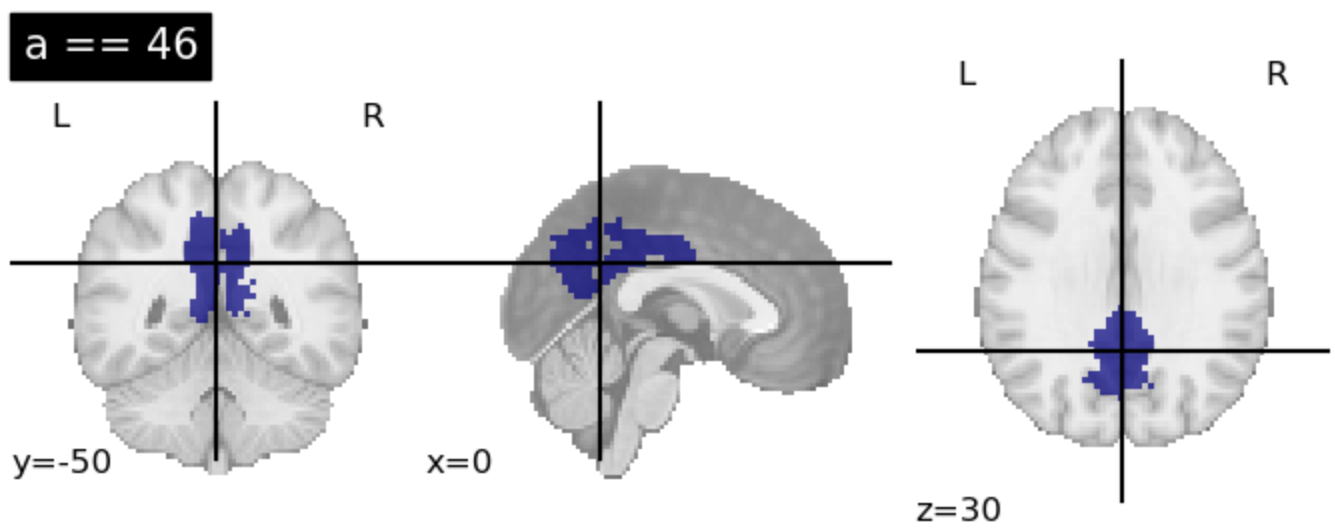

a == 41
L
R
y=-10
x=56
L
R
z=-16

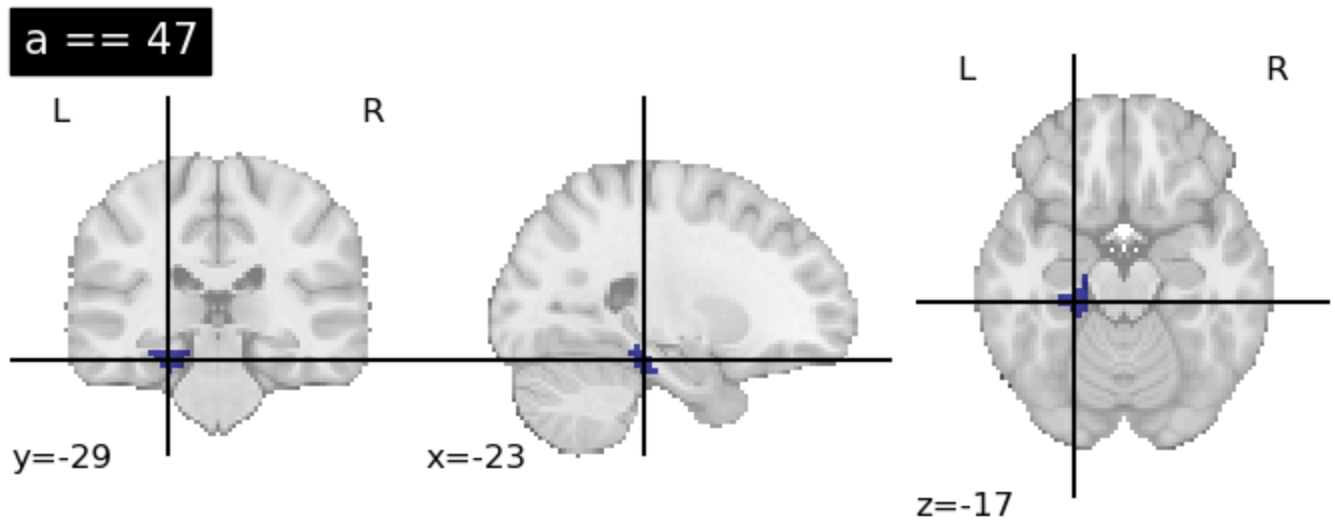

a == 42
L
R
y=-56
x=51
L
R
z=30

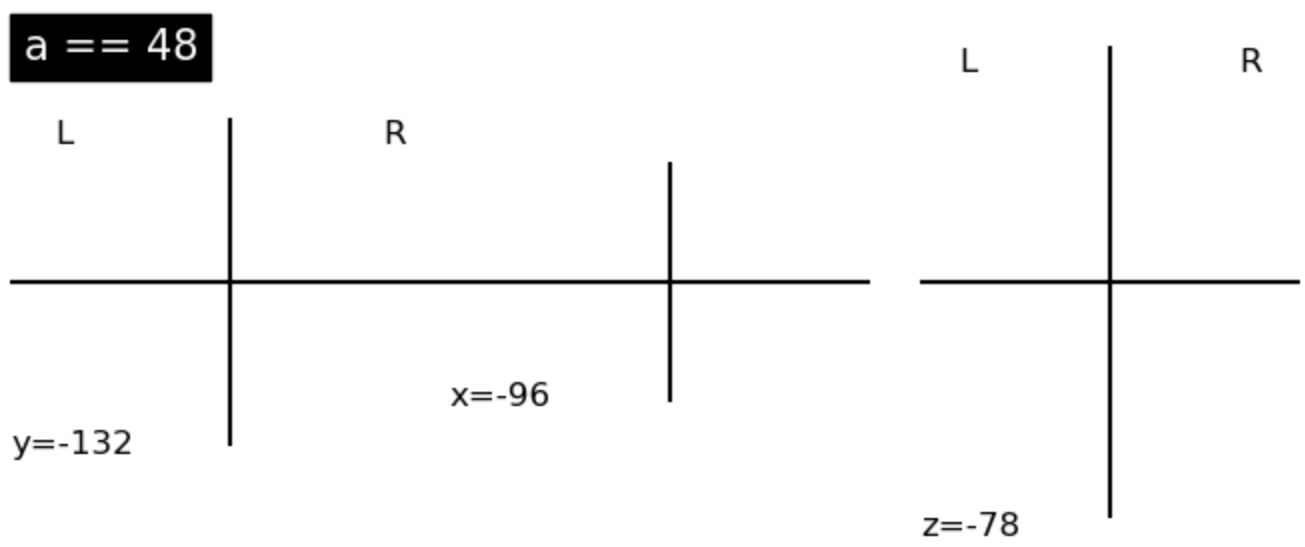

a == 43
L
R
y=29
x=45
L
R
z=-7

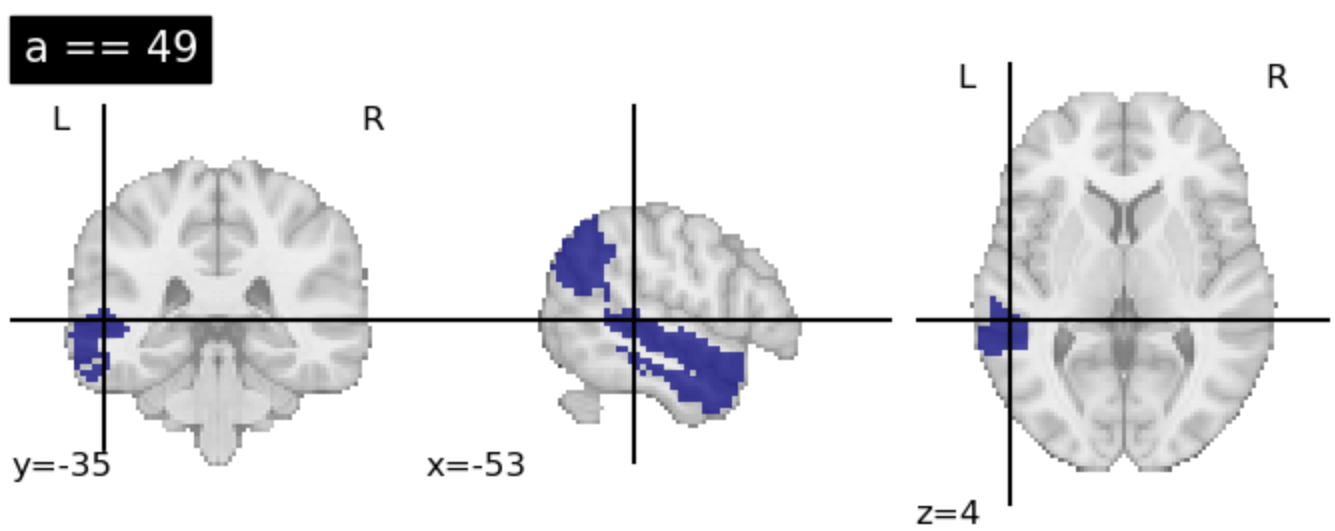

a == 44
L
R
y=40
x=-10
L
R
z=22


a == 45
L
R
y=-25
x=21
L
R
z=-19


a == 46
L
R
y=-50
x=0
L
R
z=30







Functional Connectivity Analysis

In [141…

```
# How can we estimate brain functional connectivity
# patterns from data?
from nilearn import image as nimg
from nilearn import plotting as nplot
import numpy as np
import pandas as pd
from bids import BIDSLayout
```

In [142…
```
# Use PyBIDS to parse BIDS data structure
layout = BIDSLayout(fmriprep_dir, config=['bids','derivatives'])
```

In [143…
```
# Get musical data (preprocessed, mask, and confounds file)
func_files = layout.get(subject=sub,
                        datatype='func', task='music',
                        desc='preproc',
                        space='MNI152NLin2009cAsym',
                        extension='nii.gz',
                        return_type='file')

mask_files = layout.get(subject=sub,
                        datatype='func', task='music',
                        desc='brain',
                        suffix="mask",
                        space='MNI152NLin2009cAsym',
                        extension='nii.gz',
                        return_type='file')

confound_files = layout.get(subject=sub,
                            datatype='func',
                            task='music',
                            desc='confounds',
                            extension='tsv',
                            return_type='file')
```

In [144…
```
#Load separated parcellation
parcel_file = 'resources/rois/yeo_2011/Yeo_JNeurophysiol11_MNI152/relabeled_yeo_atlas.ni
yeo_7 = nimg.load_img(parcel_file)
```

In [145…
```
# import a package from nilearn, called input_data which
# allows us to pull data using the parcellation file,
# and at the same time applying data cleaning
from nilearn import input_data
masker = input_data.NiftiLabelsMasker(labels_img=yeo_7,
                                      standardize=True,
                                      memory='nilearn_cache',
                                      verbose=1,
                                      detrend=True,
                                      low_pass = 0.08,
                                      high_pass = 0.009,
                                      t_r=3)
```

In [146…
```
# Pull the first subject's data
func_file = func_files[0]
mask_file = mask_files[0]
confound_file = confound_files[0]
```

In [147…
```
# Make confounds matrix
def extract_confounds(confound_tsv, confounds, dt=True):
    '''
    Arguments:
        confound_tsv    Full path to confounds.tsv
        confounds       A list of confounder variables to extract
        dt              Compute temporal derivatives [default = True]

    Outputs:
        confound_mat
    '''

    if dt:
        dt_names = ['{}_derivative1'.format(c) for c in confounds]
        confounds = confounds + dt_names
```

```
    #Load in data using Pandas then extract relevant columns
    confound_df = pd.read_csv(confound_tsv,delimiter='\t')
    confound_df = confound_df[confounds]


    #Convert into a matrix of values (timepoints)x(variable)
    confound_mat = confound_df.values

    #Return confound matrix
    return confound_mat
```

In [148…
```
# Drop Dummy TRs that are to be excluded from our cleaning,
# parcellation, and averaging step
# Load functional image
tr_drop = 4
func_img = nimg.load_img(func_file)

# Remove the first 4 TRs
func_img = func_img.slicer[:,:,:,tr_drop:]

# Use the above function to pull out a confound matrix
confounds = extract_confounds(confound_file,
                              ['trans_x','trans_y','trans_z',
                               'rot_x','rot_y','rot_z',
                               'global_signal',
                               'white_matter','csf'])
# Drop the first 4 rows of the confounds matrix
confounds = confounds[tr_drop:,:]
```

use the masker to perform our: - Confounds cleaning - Parcellation - Averaging within a parcel

In [149…
```
# Using the masker, apply cleaning, parcellation
# and extraction to functional data
cleaned_and_averaged_time_series = masker.fit_transform(func_img,
                                                        confounds)
cleaned_and_averaged_time_series.shape
```

```
[NiftiLabelsMasker.fit_transform] loading data from Nifti1Image('resources\rois\yeo_2011
\Yeo_JNeurophysiol11_MNI152\relabeled_yeo_atlas.nii.gz')
Resampling labels
```
Out[149]:
```
(101, 45)
```

In [150…
```
# check which ROIs are kept
print(masker.labels_)
print("Number of labels", len(masker.labels_))
```

```
[1, 2, 4, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 2
7, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 49]
Number of labels 45
```

In [151…
```
# fills in the regions that were removed with 0 values
# (for ease of use when working with multiple subjects)
# first, identify all ROIs from the original atlas
# Get the label numbers from the atlas
atlas_labels = np.unique(yeo_7.get_fdata().astype(int))

# Get number of labels that we have
NUM_LABELS = len(atlas_labels)
print(NUM_LABELS)
```

```
50
```

Now we’re going to create an array that contains: - A number of rows matching the number of timepoints - A number of columns matching the total number of regions

In [152…
```
# Remember fMRI images are of size (x,y,z,t)
# where t is the number of timepoints
num_timepoints = func_img.shape[3]

# Create an array of zeros that has the correct size
final_signal = np.zeros((num_timepoints, NUM_LABELS))

# Get regions that are kept
regions_kept = np.array(masker.labels_)

# Fill columns matching labels with signal values
final_signal[:, regions_kept] = cleaned_and_averaged_time_series

print(final_signal.shape)
```

```
(101, 50)
```

In [153…
```
# keep track of regions that was not removed my masker
valid_regions_signal = final_signal[:, regions_kept]
print(valid_regions_signal.shape)
```

```
(101, 45)
```

In [154…
```
np.array_equal(
    valid_regions_signal,
    cleaned_and_averaged_time_series)
```

Out[154]:
```
True
```

In fMRI imaging, connectivity typically refers to the correlation of the timeseries of 2 ROIs. Therefore we can calculate a full connectivity matrix by computing the correlation between all pairs of ROIs in our parcellation scheme.

In [155…
```
# Calculating Connectivity
from nilearn.connectome import ConnectivityMeasure
correlation_measure = ConnectivityMeasure(kind='correlation')
```

In [156…
```
# calculate the full correlation matrix for our parcellated data
full_correlation_matrix = correlation_measure.fit_transform([final_signal])
full_correlation_matrix.shape
```

Out[156]:
```
(1, 50, 50)
```

The result is a matrix which has:

- A number of rows matching the number of ROIs in our parcellation atlas
- A number of columns, that also matches the number of ROIs in our parcellation atlas

read this correlation matrix as follows: Suppose we wanted to know the correlation between ROI 30 and ROI 40. Then Row 30, Column 40 gives us this correlation.

In [157…
```
full_correlation_matrix[0, 43, 45]
```

Out[157]:
```
0.4139353220429633
```

Music

In [158…
```
# now apply it to every subject
# First we're going to create some empty lists to store all our data in
```

```
pooled_subjects = []
ctrl_subjects = []
mdd_subjects = []

# Which confound variables should we use?
confound_variables = ['trans_x','trans_y','trans_z',
                      'rot_x','rot_y','rot_z',
                      'global_signal',
                      'white_matter','csf']
# get the list of subjects
subjects = layout.get_subjects()
for sub in subjects:

    #Get the functional file for the subject (MNI space)
    func_files = layout.get(subject=sub,
                            datatype='func', task='music',
                            desc='preproc',
                            extension="nii.gz",
                            return_type='file')

    #Get the confounds file for the subject (MNI space)
    confound_files = layout.get(subject=sub, datatype='func',
                             task='music',
                             desc='confounds',
                             extension='tsv',
                             return_type='file')

    for i in range(len(func_files)):
        #Load the functional file in
        func_img = nimg.load_img(func_files[i])

        #Drop the first 4 TRs
        func_img = func_img.slicer[:,:,:,tr_drop:]

        #Extract the confound variables using the function
        confounds = extract_confounds(confound_files[i],
                                      confound_variables)

        #Drop the first 4 rows from the confound matrix
        #Which rows and columns should we keep?
        confounds = confounds[tr_drop:,:]

        #Apply the parcellation + cleaning to our data
        #What function of masker is used to clean and average data?
        time_series = masker.fit_transform(func_img, confounds)

        # fill the drop ROIs
        # Remember fMRI images are of size (x,y,z,t)
        # where t is the number of timepoints
        num_timepoints = func_img.shape[3]

        # Create an array of zeros that has the correct size
        final_signal = np.zeros((num_timepoints, NUM_LABELS))

        # Get regions that are kept
        regions_kept = np.array(masker.labels_)

        # Fill columns matching labels with signal values
        final_signal[:, regions_kept] = time_series

        #This collects a list of all subjects
        pooled_subjects.append(final_signal)

        #If the subject ID starts with a "control" then they are control
        if sub.startswith('control'):
            ctrl_subjects.append(final_signal)
```

```
        #If the subject ID starts with a "mdd" then they are mdd
        if sub.startswith('mdd'):
            mdd_subjects.append(final_signal)
```

```
[NiftiLabelsMasker.fit_transform] loading data from Nifti1Image('resources\rois\yeo_2011
\Yeo_JNeurophysiol11_MNI152\relabeled_yeo_atlas.nii.gz')
[NiftiLabelsMasker.fit_transform] loading data from Nifti1Image('resources\rois\yeo_2011
\Yeo_JNeurophysiol11_MNI152\relabeled_yeo_atlas.nii.gz')
[NiftiLabelsMasker.fit_transform] loading data from Nifti1Image('resources\rois\yeo_2011
\Yeo_JNeurophysiol11_MNI152\relabeled_yeo_atlas.nii.gz')
[NiftiLabelsMasker.fit_transform] loading data from Nifti1Image('resources\rois\yeo_2011
\Yeo_JNeurophysiol11_MNI152\relabeled_yeo_atlas.nii.gz')
[NiftiLabelsMasker.fit_transform] loading data from Nifti1Image('resources\rois\yeo_2011
\Yeo_JNeurophysiol11_MNI152\relabeled_yeo_atlas.nii.gz')
[NiftiLabelsMasker.fit_transform] loading data from Nifti1Image('resources\rois\yeo_2011
\Yeo_JNeurophysiol11_MNI152\relabeled_yeo_atlas.nii.gz')
[NiftiLabelsMasker.fit_transform] loading data from Nifti1Image('resources\rois\yeo_2011
\Yeo_JNeurophysiol11_MNI152\relabeled_yeo_atlas.nii.gz')
[NiftiLabelsMasker.fit_transform] loading data from Nifti1Image('resources\rois\yeo_2011
\Yeo_JNeurophysiol11_MNI152\relabeled_yeo_atlas.nii.gz')
[NiftiLabelsMasker.fit_transform] loading data from Nifti1Image('resources\rois\yeo_2011
\Yeo_JNeurophysiol11_MNI152\relabeled_yeo_atlas.nii.gz')
[NiftiLabelsMasker.fit_transform] loading data from Nifti1Image('resources\rois\yeo_2011
\Yeo_JNeurophysiol11_MNI152\relabeled_yeo_atlas.nii.gz')
[NiftiLabelsMasker.fit_transform] loading data from Nifti1Image('resources\rois\yeo_2011
\Yeo_JNeurophysiol11_MNI152\relabeled_yeo_atlas.nii.gz')
[NiftiLabelsMasker.fit_transform] loading data from Nifti1Image('resources\rois\yeo_2011
\Yeo_JNeurophysiol11_MNI152\relabeled_yeo_atlas.nii.gz')
```

In [159…

```
# calculate the full correlation matrix for all data
# (correlation_measure works on the list as well)
ctrl_correlation_matrices = correlation_measure.fit_transform(ctrl_subjects)
mdd_correlation_matrices = correlation_measure.fit_transform(mdd_subjects)
```

Visualizing Correlation Matrices and Group Differences

In [160…

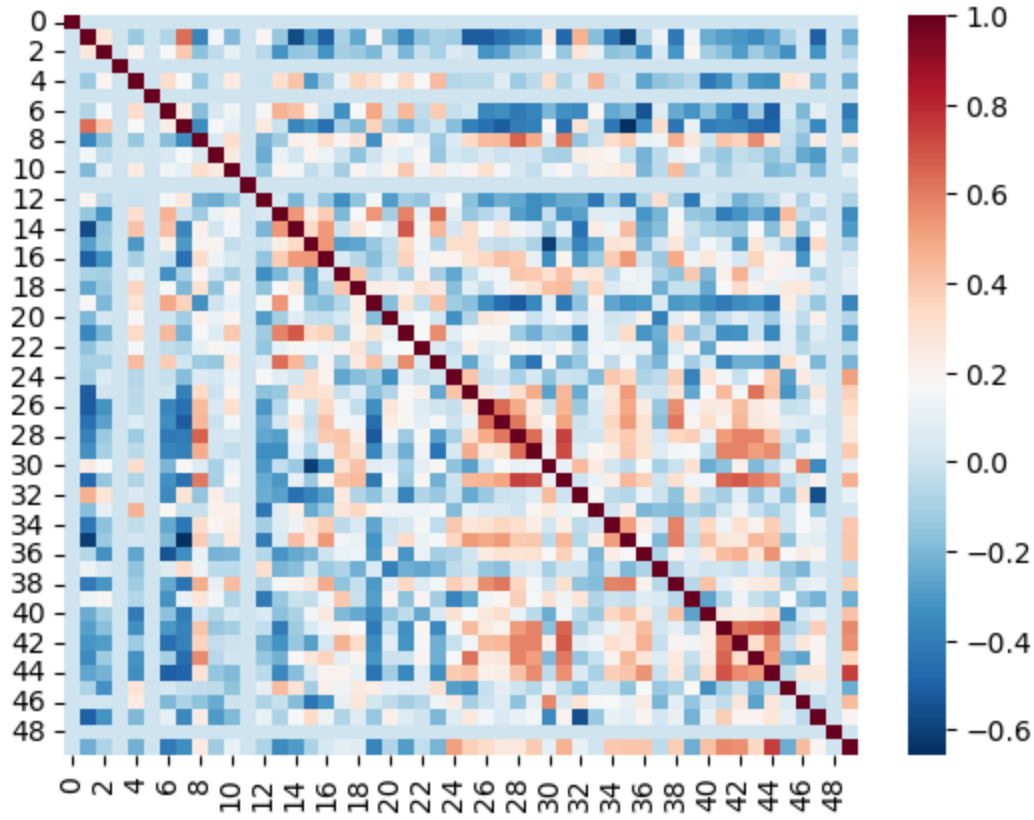
```
import seaborn as sns
import matplotlib.pyplot as plt
```

In [161…

```
sns.heatmap(ctrl_correlation_matrices[0], cmap='RdBu_r')
```

Out[161]:

```
<AxesSubplot:>
```



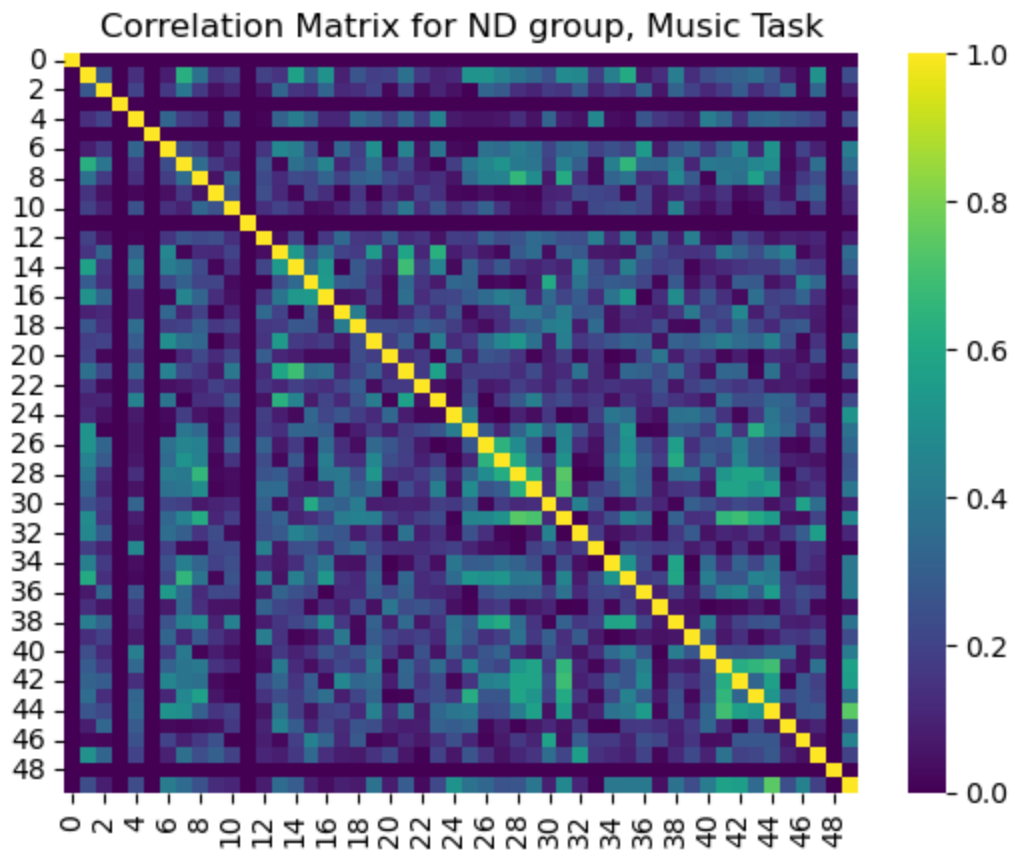
to make it more apparent:

- Taken the absolute value of our correlations so that the 0's are the darkest color
- Used a different color scheme

The dark line is basically the ROIs that were drop (dropped because it contains no voxels)

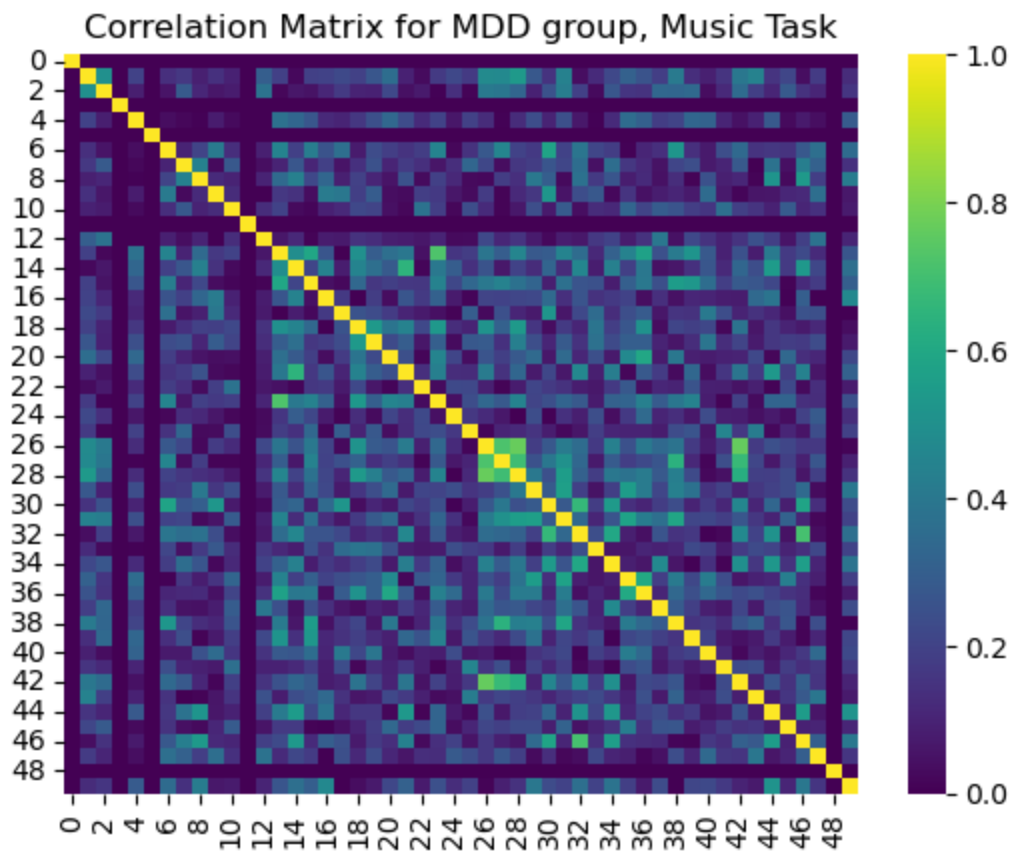
In [162…
```
sns.heatmap(np.abs(ctrl_correlation_matrices[0]), cmap='viridis').set_title('Correlation
```

Out[162]:
```
Text(0.5, 1.0, 'Correlation Matrix for ND group, Music Task')
```



In [163…
```
sns.heatmap(np.abs(mdd_correlation_matrices[0]), cmap='viridis').set_title('Correlation
```

Out[163]:
```
Text(0.5, 1.0, 'Correlation Matrix for MDD group, Music Task')
```



In [164…
```
print(ctrl_correlation_matrices.shape)
```

```
(6, 50, 50)
```

In [165…
```
# pull out just the correlation values between ROI 44 and 41 across all our subjects
```

```
ctrl_roi_vec = ctrl_correlation_matrices[:,44,41]
mdd_roi_vec = mdd_correlation_matrices[:,44,41]
```

In [166…

```
# arrange this data into a table
#Create control dataframe
ctrl_df = pd.DataFrame(data={'AC_ACC_corr':ctrl_roi_vec,
                             'group':'control'})
ctrl_df.head()
```

Out[166]:

	AC_ACC_corr	group
0	0.692324	control
1	0.499350	control
2	0.421852	control
3	0.636363	control
4	0.448103	control

In [167…

```
# Create the mdd dataframe
mdd_df = pd.DataFrame(data={'AC_ACC_corr':mdd_roi_vec,
                            'group' : 'mdd'})
mdd_df.head()
```

Out[167]:

	AC_ACC_corr	group
0	-0.068150	mdd
1	0.369329	mdd
2	0.305971	mdd
3	0.294117	mdd
4	0.398849	mdd

In [168…

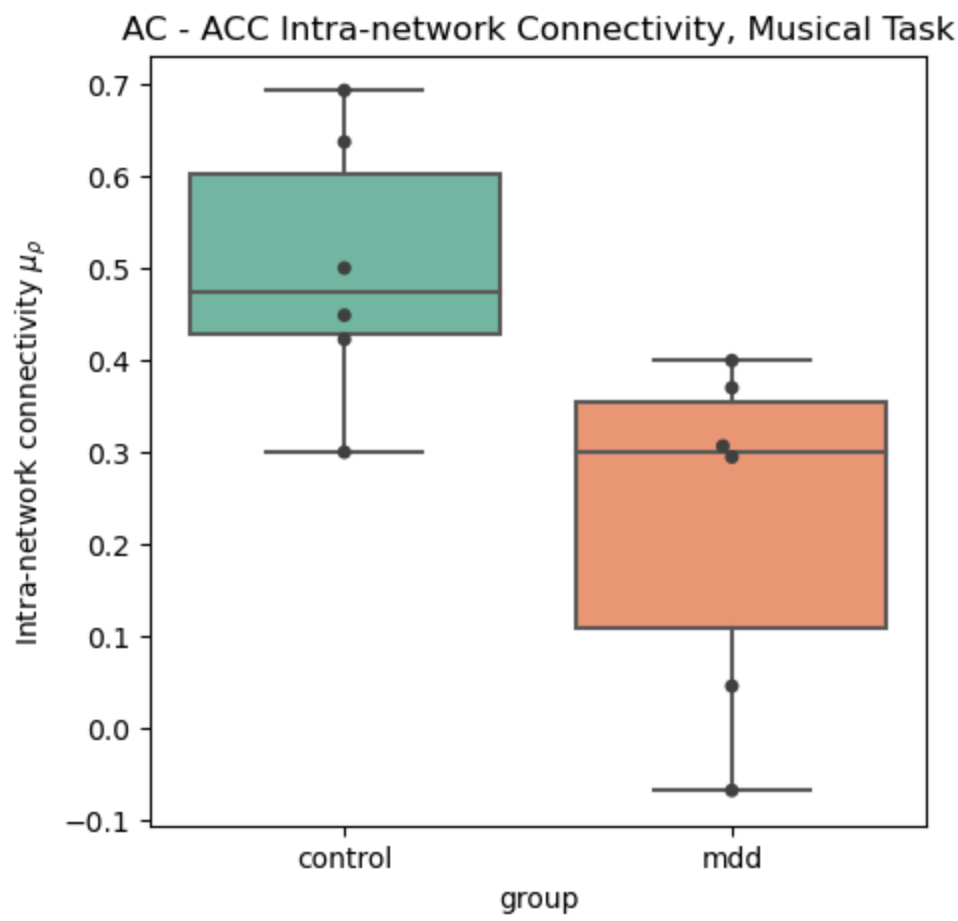
```
# For visualization stack the two tables together
#Stack the two dataframes together
df_music = pd.concat([ctrl_df,mdd_df], ignore_index=True)

# Show some random samples from dataframe
# df.sample(n=7)

df_music
```

Out[168]:

	AC_ACC_corr	group
0	0.692324	control
1	0.499350	control
2	0.421852	control
3	0.636363	control
4	0.448103	control
5	0.299380	control
6	-0.068150	mdd
7	0.369329	mdd
8	0.305971	mdd

9	0.294117	mdd
10	0.398849	mdd
11	0.045207	mdd

In [169…
```
# Visualize results

# Create a figure canvas of equal width and height
plot = plt.figure(figsize=(5,5))

# Create a box plot, with the x-axis as group
# the y-axis as the correlation value
ax = sns.boxplot(x='group',y='AC_ACC_corr',
                 data=df_music,palette='Set2')

# Create a "swarmplot" as well
ax = sns.swarmplot(x='group',y='AC_ACC_corr',
                   data=df_music,color='0.25')

# Set the title and labels of the figure
ax.set_title('AC - ACC Intra-network Connectivity, Musical Task')
ax.set_ylabel(r'Intra-network connectivity $\mu_\rho$')

plt.show()
```



In [170…
```
# pull out just the correlation values between
# ROI 44 and 49 across all our subjects
ctrl_roi_vec = ctrl_correlation_matrices[:,44,49]
mdd_roi_vec = mdd_correlation_matrices[:,44,49]
```

In [171…
```
# arrange this data into a table
#Create control dataframe
ctrl_df = pd.DataFrame(data={'AC_ACC_corr':ctrl_roi_vec,
```

```
                                        'group':'control'})
ctrl_df.head()
```

Out[171]:

	AC_ACC_corr	group
0	0.745819	control
1	0.550316	control
2	0.676431	control
3	0.454386	control
4	0.400613	control

In [172…

```
# Create the mdd dataframe
mdd_df = pd.DataFrame(data={'AC_ACC_corr':mdd_roi_vec,
                            'group' : 'mdd'})
mdd_df.head()
```

Out[172]:

	AC_ACC_corr	group
0	0.512664	mdd
1	0.758968	mdd
2	0.516627	mdd
3	0.694276	mdd
4	0.707918	mdd

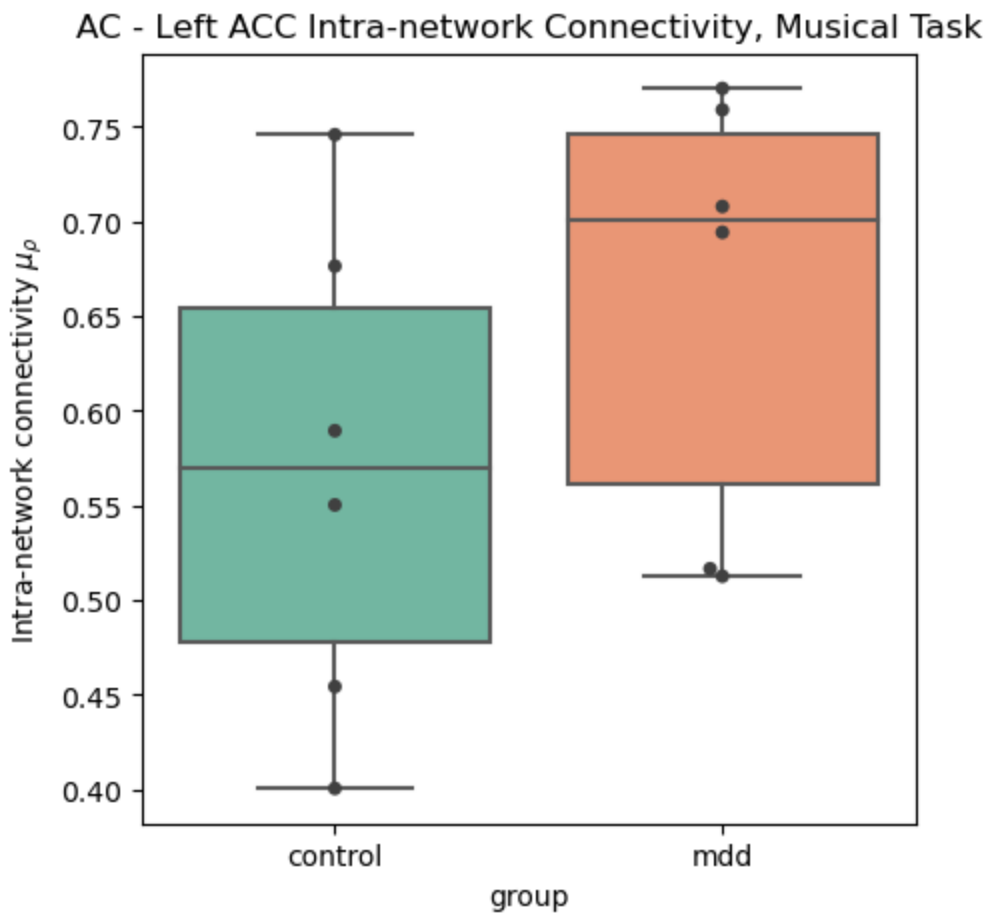
In [173…

```
# For visualization stack the two tables together
#Stack the two dataframes together
df_music = pd.concat([ctrl_df,mdd_df], ignore_index=True)

# Show some random samples from dataframe
# df.sample(n=7)

df_music
```

Out[173]:

	AC_ACC_corr	group
0	0.745819	control
1	0.550316	control
2	0.676431	control
3	0.454386	control
4	0.400613	control
5	0.589507	control
6	0.512664	mdd
7	0.758968	mdd
8	0.516627	mdd
9	0.694276	mdd
10	0.707918	mdd
11	0.770246	mdd

In [174…
```
# Visualize results

# Create a figure canvas of equal width and height
plot = plt.figure(figsize=(5,5))

# Create a box plot, with the x-axis as group
# the y-axis as the correlation value
ax = sns.boxplot(x='group',y='AC_ACC_corr',
                 data=df_music,palette='Set2')

# Create a "swarmplot" as well
ax = sns.swarmplot(x='group',y='AC_ACC_corr',
                   data=df_music,color='0.25')

# Set the title and labels of the figure
ax.set_title('AC - Left ACC Intra-network Connectivity, Musical Task')
ax.set_ylabel(r'Intra-network connectivity $\mu_\rho$')

plt.show()
```



Non-Music

In [175…
```
# now apply it to every subject for non-musical task
# First we're going to create some empty lists to store all our data in
pooled_subjects = []
ctrl_subjects = []
mdd_subjects = []

# Which confound variables should we use?
confound_variables = ['trans_x','trans_y','trans_z',
                                'rot_x','rot_y','rot_z',
                                'global_signal',
                                'white_matter','csf']
```

```
# get the list of subjects
subjects = layout.get_subjects()
for sub in subjects:

    #Get the functional file for the subject (MNI space)
    func_files = layout.get(subject=sub,
                            datatype='func', task='nonmusic',
                            desc='preproc',
                            extension="nii.gz",
                            return_type='file')

    #Get the confounds file for the subject (MNI space)
    confound_files = layout.get(subject=sub, datatype='func',
                                task='nonmusic',
                                desc='confounds',
                                extension='tsv',
                                return_type='file')

    for i in range(len(func_files)):
        #Load the functional file in
        func_img = nimg.load_img(func_files[i])

        #Drop the first 4 TRs
        func_img = func_img.slicer[:,:,:,tr_drop:]

        #Extract the confound variables using the function
        confounds = extract_confounds(confound_files[i],
                                      confound_variables)

        #Drop the first 4 rows from the confound matrix
        #Which rows and columns should we keep?
        confounds = confounds[tr_drop:,:]

        #Apply the parcellation + cleaning to our data
        #What function of masker is used to clean and average data?
        time_series = masker.fit_transform(func_img, confounds)

        # fill the drop ROIs
        # Remember fMRI images are of size (x,y,z,t)
        # where t is the number of timepoints
        num_timepoints = func_img.shape[3]

        # Create an array of zeros that has the correct size
        final_signal = np.zeros((num_timepoints, NUM_LABELS))

        # Get regions that are kept
        regions_kept = np.array(masker.labels_)

        # Fill columns matching labels with signal values
        final_signal[:, regions_kept] = time_series

        #This collects a list of all subjects
        pooled_subjects.append(final_signal)

        #If the subject ID starts with a "control" then they are control
        if sub.startswith('control'):
            ctrl_subjects.append(final_signal)
        #If the subject ID starts with a "mdd" then they are mdd
        if sub.startswith('mdd'):
            mdd_subjects.append(final_signal)
```

```
[NiftiLabelsMasker.fit_transform] loading data from Nifti1Image('resources\rois\yeo_2011
\Yeo_JNeurophysiol11_MNI152\relabeled_yeo_atlas.nii.gz')
[NiftiLabelsMasker.fit_transform] loading data from Nifti1Image('resources\rois\yeo_2011
\Yeo_JNeurophysiol11_MNI152\relabeled_yeo_atlas.nii.gz')
[NiftiLabelsMasker.fit_transform] loading data from Nifti1Image('resources\rois\yeo_2011
```

```
\Yeo_JNeurophysiol11_MNI152\relabeled_yeo_atlas.nii.gz')
[NiftiLabelsMasker.fit_transform] loading data from Nifti1Image('resources\rois\yeo_2011
\Yeo_JNeurophysiol11_MNI152\relabeled_yeo_atlas.nii.gz')
[NiftiLabelsMasker.fit_transform] loading data from Nifti1Image('resources\rois\yeo_2011
\Yeo_JNeurophysiol11_MNI152\relabeled_yeo_atlas.nii.gz')
[NiftiLabelsMasker.fit_transform] loading data from Nifti1Image('resources\rois\yeo_2011
\Yeo_JNeurophysiol11_MNI152\relabeled_yeo_atlas.nii.gz')
[NiftiLabelsMasker.fit_transform] loading data from Nifti1Image('resources\rois\yeo_2011
\Yeo_JNeurophysiol11_MNI152\relabeled_yeo_atlas.nii.gz')
[NiftiLabelsMasker.fit_transform] loading data from Nifti1Image('resources\rois\yeo_2011
\Yeo_JNeurophysiol11_MNI152\relabeled_yeo_atlas.nii.gz')
```

In [176…

```
# calculate the full correlation matrix for all data
# (correlation_measure works on the list as well)
ctrl_correlation_matrices = correlation_measure.fit_transform(ctrl_subjects)
mdd_correlation_matrices = correlation_measure.fit_transform(mdd_subjects)
```

Visualizing Correlation Matrices and Group Differences

In [177…

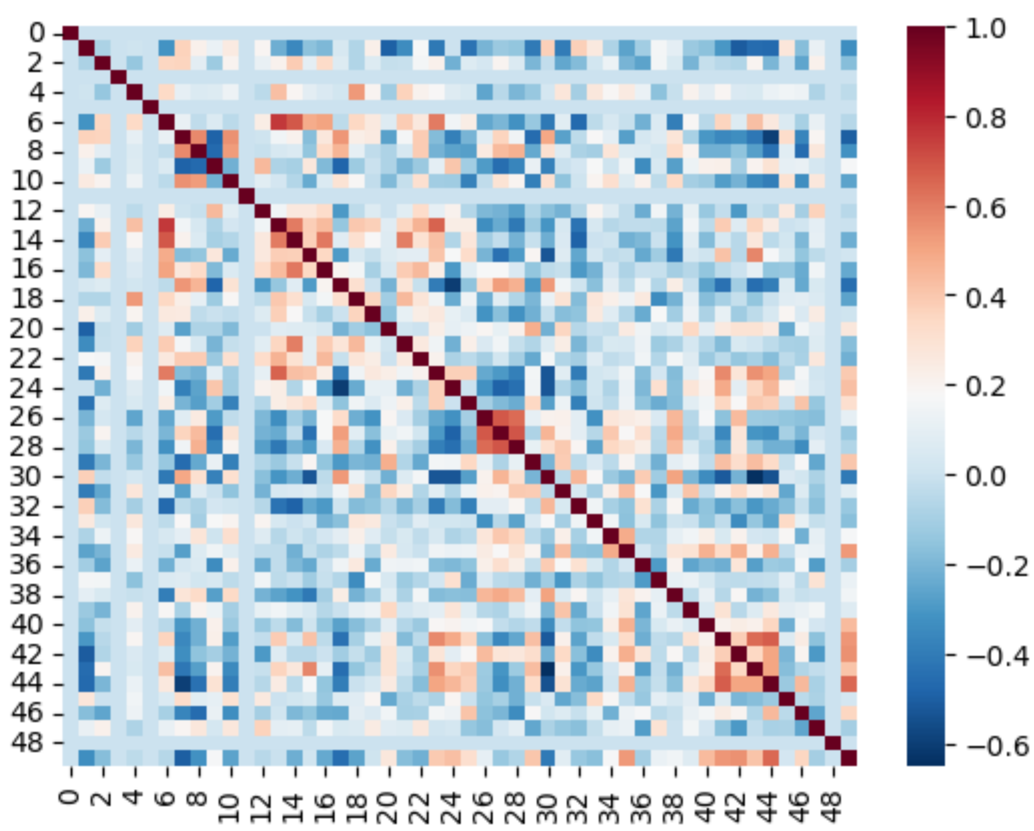
```
import seaborn as sns
import matplotlib.pyplot as plt
```

In [178…

```
sns.heatmap(ctrl_correlation_matrices[0], cmap='RdBu_r')
```

Out[178]:

```
<AxesSubplot:>
```



to make it more apparent:

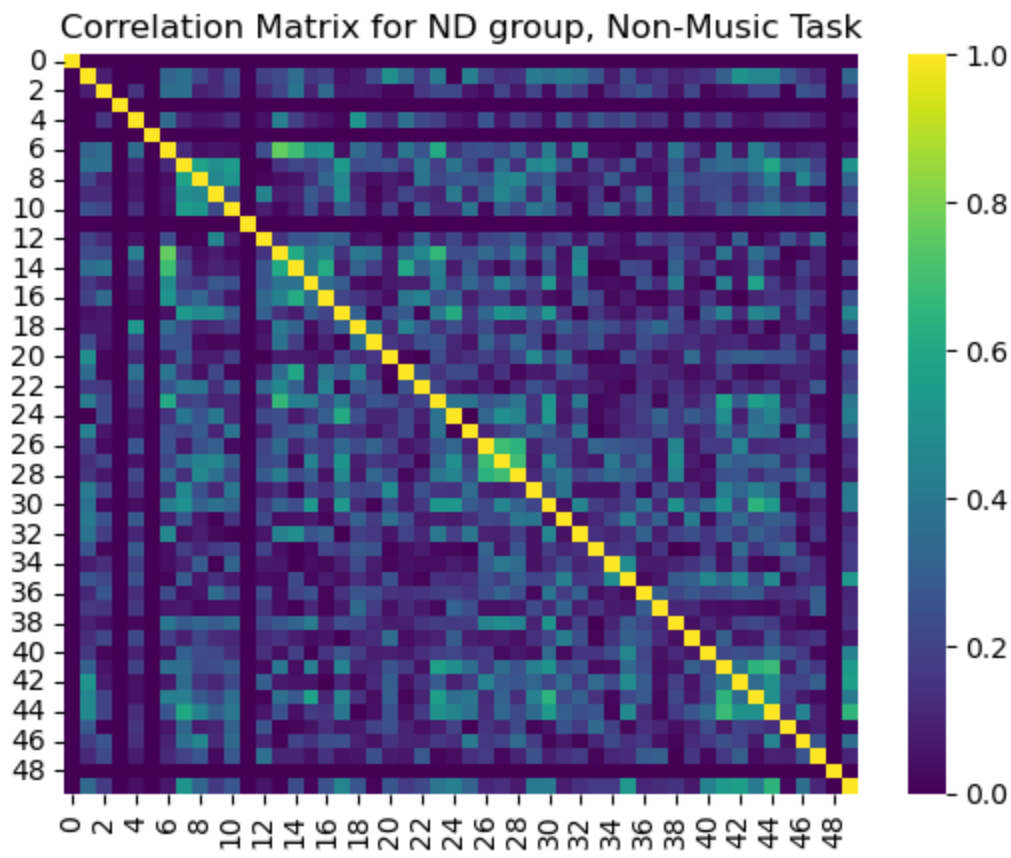
- Taken the absolute value of our correlations so that the 0's are the darkest color
- Used a different color scheme

The dark line is basically the ROIs that were drop (dropped because it contains no voxels)

In [179…

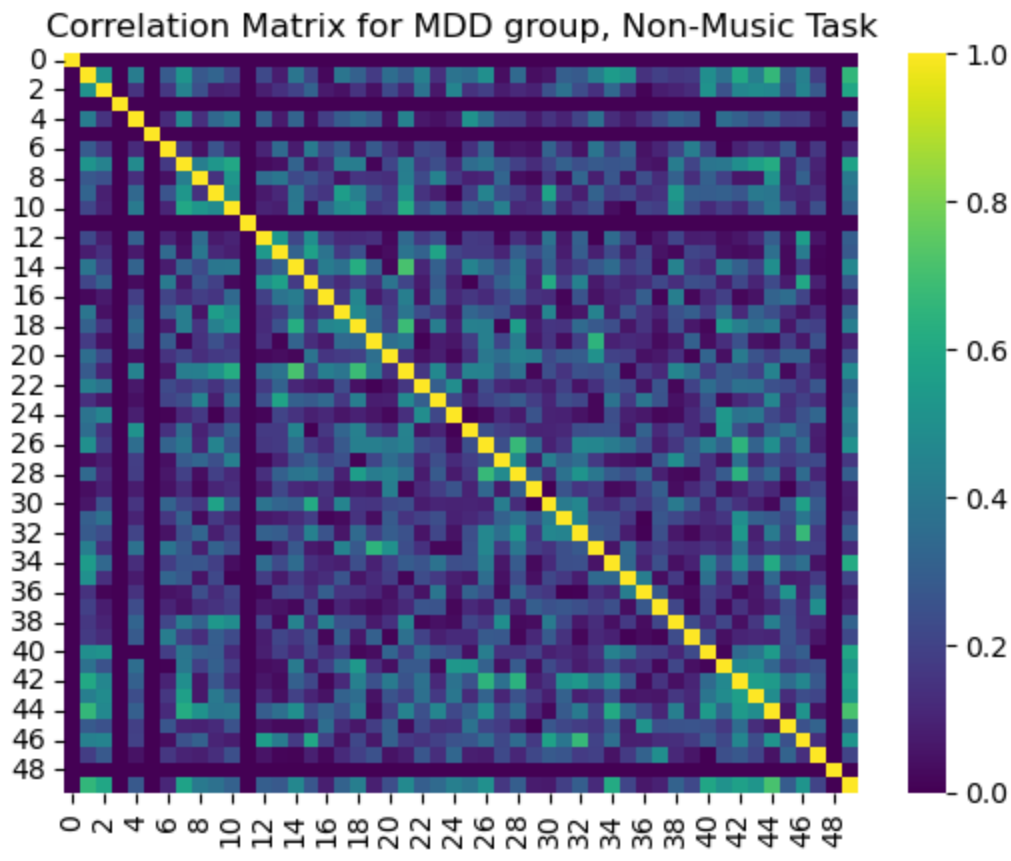
```
sns.heatmap(np.abs(ctrl_correlation_matrices[0]), cmap='viridis').set_title('Correlation
```

Out[179]: Text(0.5, 1.0, 'Correlation Matrix for ND group, Non-Music Task')



In [180…
```
sns.heatmap(np.abs(mdd_correlation_matrices[0]), cmap='viridis').set_title('Correlation
```

Out[180]: Text(0.5, 1.0, 'Correlation Matrix for MDD group, Non-Music Task')



In [181…
```
print(ctrl_correlation_matrices.shape)
```

```
(4, 50, 50)
```

In [182…
```
# pull out just the correlation values between
# ROI 44 and 41 across all our subjects
ctrl_roi_vec = ctrl_correlation_matrices[:,44,41]
mdd_roi_vec = mdd_correlation_matrices[:,44,41]
```

In [183…
```
# arrange this data into a table
#Create control dataframe
ctrl_df = pd.DataFrame(data={'AC_ACC_corr':ctrl_roi_vec, 'group':'control'})
ctrl_df.head()
```

Out[183]:

	AC_ACC_corr	group
0	0.676406	control
1	0.591649	control
2	0.331425	control
3	0.518895	control

In [184…
```
# Create the mdd dataframe
mdd_df = pd.DataFrame(data={'AC_ACC_corr':mdd_roi_vec, 'group' : 'mdd'})
mdd_df.head()
```

Out[184]:

	AC_ACC_corr	group
0	0.320510	mdd
1	0.112568	mdd
2	0.172342	mdd
3	0.496927	mdd

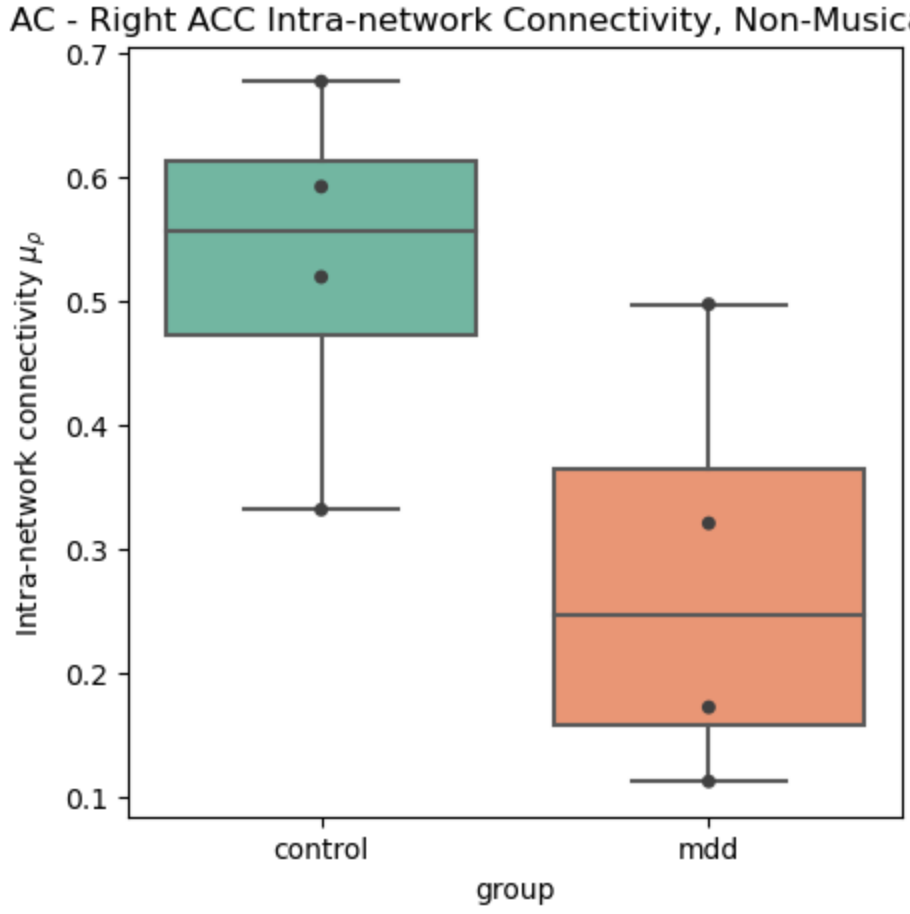
In [185…
```
# For visualization stack the two tables together
#Stack the two dataframes together
df_nonmusic = pd.concat([ctrl_df,mdd_df], ignore_index=True)

# Show some random samples from dataframe
# df.sample(n=7)

df_nonmusic
```

Out[185]:

	AC_ACC_corr	group
0	0.676406	control
1	0.591649	control
2	0.331425	control
3	0.518895	control
4	0.320510	mdd
5	0.112568	mdd
6	0.172342	mdd
7	0.496927	mdd

In [186…
```
# Visualize results

# Create a figure canvas of equal width and height
plot = plt.figure(figsize=(5,5))
```

```
# Create a box plot, with the x-axis as group
# the y-axis as the correlation value
ax = sns.boxplot(x='group',y='AC_ACC_corr',data=df_nonmusic,palette='Set2')

# Create a "swarmplot" as well
ax = sns.swarmplot(x='group',y='AC_ACC_corr',data=df_nonmusic,color='0.25')

# Set the title and labels of the figure
ax.set_title('AC - Right ACC Intra-network Connectivity, Non-Musical Task')
ax.set_ylabel(r'Intra-network connectivity $\mu_\rho$')

plt.show()
```



In [187…
```
# pull out just the correlation values between ROI 44 and 49 across all our subjects
ctrl_roi_vec = ctrl_correlation_matrices[:,44,49]
mdd_roi_vec = mdd_correlation_matrices[:,44,49]
```

In [188…
```
# arrange this data into a table
#Create control dataframe
ctrl_df = pd.DataFrame(data={'AC_ACC_corr':ctrl_roi_vec, 'group':'control'})
ctrl_df.head()
```

Out[188]:

	AC_ACC_corr	group
0	0.647581	control
1	0.672488	control
2	0.367679	control
3	0.469284	control

In [189…
```
# Create the mdd dataframe
mdd_df = pd.DataFrame(data={'AC_ACC_corr':mdd_roi_vec, 'group' : 'mdd'})
```

```
mdd_df.head()
```

Out[189]:

	AC_ACC_corr	group
0	0.708775	mdd
1	0.448094	mdd
2	0.691701	mdd
3	0.576144	mdd

In [190…

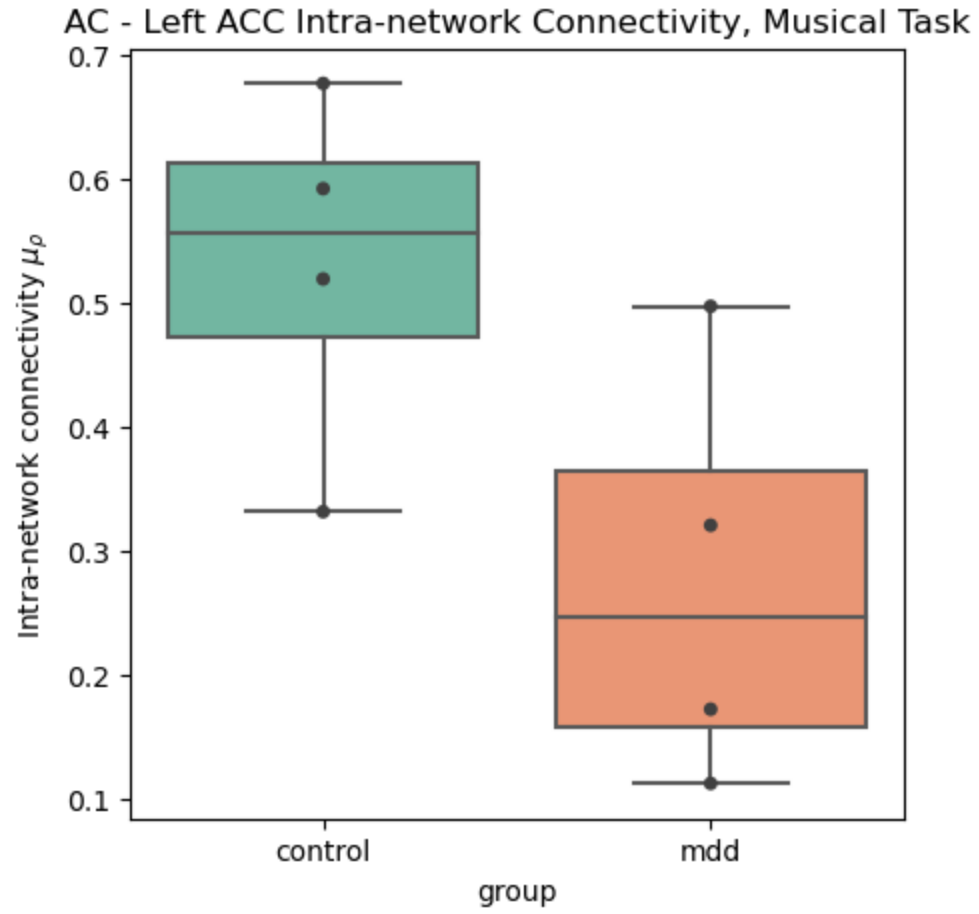
```
# For visualization stack the two tables together
#Stack the two dataframes together
df_music = pd.concat([ctrl_df,mdd_df], ignore_index=True)

# Show some random samples from dataframe
# df.sample(n=7)

df_music
```

Out[190]:

	AC_ACC_corr	group
0	0.647581	control
1	0.672488	control
2	0.367679	control
3	0.469284	control
4	0.708775	mdd
5	0.448094	mdd
6	0.691701	mdd
7	0.576144	mdd

In [191…

```
# Visualize results

# Create a figure canvas of equal width and height
plot = plt.figure(figsize=(5,5))

# Create a box plot, with the x-axis as group
# the y-axis as the correlation value
ax = sns.boxplot(x='group',y='AC_ACC_corr',
                 data=df_nonmusic,palette='Set2')

# Create a "swarmplot" as well
ax = sns.swarmplot(x='group',y='AC_ACC_corr',
                   data=df_nonmusic,color='0.25')

# Set the title and labels of the figure
ax.set_title('AC - Left ACC Intra-network Connectivity, Musical Task')
ax.set_ylabel(r'Intra-network connectivity $\mu_\rho$')

plt.show()
```



In [192…
```
from nilearn import plotting
coords = plotting.find_parcellation_cut_coords(yeo_7)
```

In [193…
```
# to match the dimension of the matrix, get coordinates
coords = np.append(coords, [[0, 0, 0]], axis = 0)
```

In [194…
```
coords.shape
```

Out[194]:
```
(50, 3)
```

In [195…
```
coords
```

Out[195]:
```
array([[-22.26254905, -77.77055908,   3.67973072],
       [ 24.27819549, -55.81203008,  57.0112782 ],
       [-14.        , -34.        ,  -4.        ],
       [-30.91803279, -23.72131148, -21.21311475],
       [-34.66666667, -24.22222222, -18.55555556],
       [-36.61613048, -21.41024624,  40.9779725 ],
       [-35.23847176, -56.43335729,  32.62734976],
       [ 37.00391294,   2.00497269,  44.26371566],
       [ -5.7       , -78.5       ,  39.3       ],
       [-33.93000904,  -0.87505134,  49.32588516],
       [-26.81818182, -32.45454545, -20.18181818],
       [-33.79573171, -27.75914634, -20.81402439],
       [ 60.40585774, -32.8252056 ,  23.5296494 ],
       [ 43.63453261,   8.79599382,   1.43665158],
       [ 52.13052012,   0.67419038,  46.31501472],
       [ 52.37285491,  39.00936037,   1.98439938],
       [ 30.01151742,  44.88252232,  29.23869853],
       [ -7.96571281,  -6.81934777,  47.49587523],
       [-29.63331371,  41.8120463 ,  31.32313125],
       [-26.05511811,  35.94488189, -11.51181102],
       [-42.22200293,   5.60183828,   2.38344423],
       [-46.67572816,  -2.49902913,  52.42718447],
```

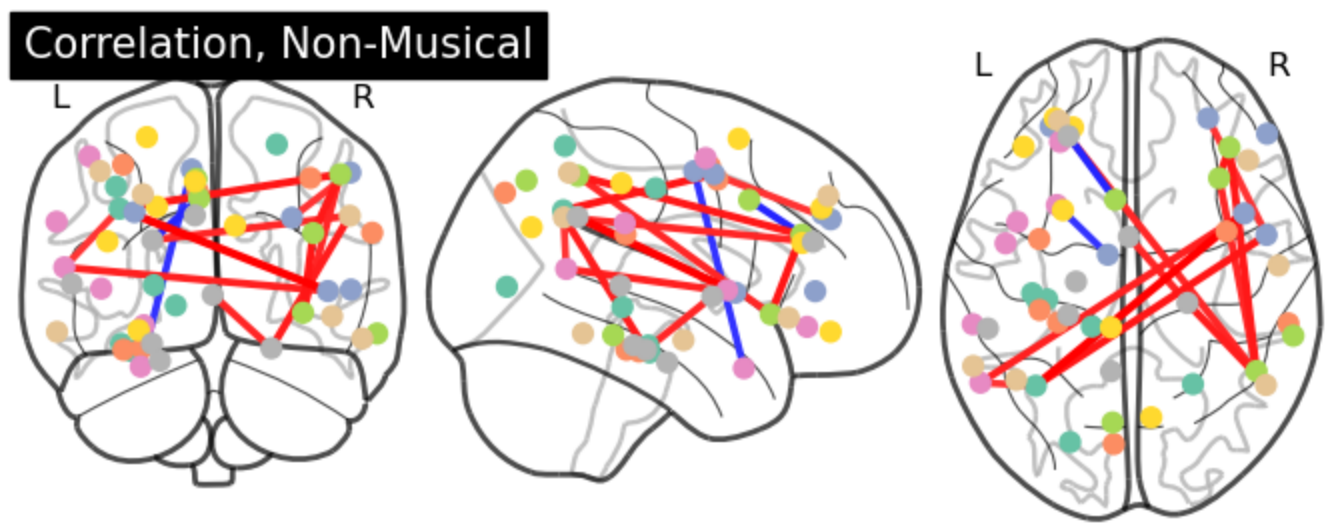
```
[-59.1147997 , -33.11461424,  27.69751484],
[-56.19502618, -55.26570681,  10.42931937],
[-27.25500787,  12.01417075, -27.1632479 ],
[ 62.29397513, -37.48699725, -14.6443224 ],
[ 48.32603441, -50.84473198,  45.68071947],
[ 38.09182412,  33.80404764,  23.39124205],
[ 34.20024198,  22.18269812,  -6.753781  ],
[ -6.15697036, -70.35199415,  43.91364801],
[ -5.14627524,  14.00763618,  36.55506533],
[  8.5       , -68.4       ,  26.2       ],
[ -6.76635514, -34.34579439,  42.87383178],
[-24.93019344,  10.03784693,  59.751612  ],
[-39.85846042,  34.0926908 ,  20.24974472],
[-28.0564242 ,  44.83922502, -13.36573759],
[-21.14545455,  41.41818182,  33.41818182],
[-42.65268759, -54.19660895,  46.88032107],
[-26.46153846,  43.92307692,  38.07692308],
[-58.95056964, -48.94891584, -13.99797868],
[ 56.64373082, -10.88827114, -16.53342525],
[ 51.91297659, -56.14983278,  29.96782609],
[ 45.20464016,  28.94706611,  -8.08199122],
[-22.87545889,  38.13783247,  20.71047782],
[ 22.20175439, -25.08947368, -20.21929825],
[ -6.75541796, -50.93873541,  30.08359133],
[-22.99221588, -29.66632071, -18.47327452],
[-20.        , -17.        , -25.        ],
[-53.40107239, -34.76639857,   4.10443849],
[  0.        ,   0.        ,   0.        ]])
```

In [196…

```
plotting.plot_connectome(ctrl_correlation_matrices[0], coords,
                         edge_threshold="99%",
                         title='Correlation, Non-Musical')
```

Out[196]:

```
<nilearn.plotting.displays._projectors.OrthoProjector at 0x2c220d40880>
```

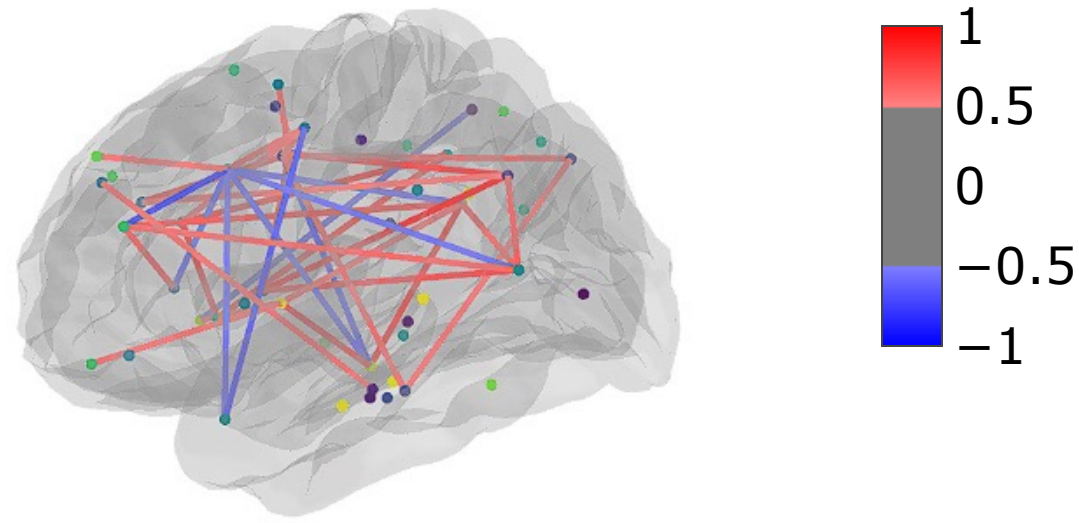


In [197…

```
view = plotting.view_connectome(ctrl_correlation_matrices[0], coords,
                                edge_threshold="95%",
                                 title='Correlation, Non-Musical')
# In a Jupyter notebook, if ``view`` is the output of a cell, it will
# be displayed below the cell
view
```

Out[197]:

Correlation, Non-Musical



view: Left ▼