Assignment 5

NSEC 3505

Working with MRI Data

This assignment is based on the DataCamp lesson, Biomedical Image Analysis in Python. To complete this assignment, you can rely on chapters 1 and 2 from that lesson. I suggest that you download the slides from those chapters and refer to them as you work through this lesson.

First we'll load all the packages we'll need for this assignment:

In [1]:
```
import imageio as iio
import scipy.ndimage as ndi
import scipy.stats
import nibabel as nib
import numpy as np
import matplotlib.pyplot as plt
```

Q 1

Use the `iio.imread` function to load in the DICOM file named `IM-0004-0096.dcm` from the `data` subfolder. Assign the object to `brain_img`

Note that although not necessary (here), it's good practice to pass two arguments to `.imread()`: the name of the image, and the format. For DICOM images, the second argument would be `"DICOM"`

In [2]:
```
brain_img = iio.imread("data/IM-0004-0096.dcm" , "DICOM")
```

Q 2

As you learned in the lesson, you can view the metadata for a DICOM file, which is stored in the `.meta` property of the object. Show all the metadata.

In [3]:
```
print(brain_img.meta.keys())
```

Out[3]:
```
odict_keys(['TransferSyntaxUID', 'SOPClassUID', 'SOPInstanceUID', 'StudyDate',
'SeriesDate', 'AcquisitionDate', 'ContentDate', 'StudyTime', 'SeriesTime',
'AcquisitionTime', 'ContentTime', 'Modality', 'Manufacturer', 'InstitutionName',
'SeriesDescription', 'PatientName', 'PatientID', 'PatientBirthDate', 'PatientSex',
'PatientAge', 'PatientWeight', 'SliceSpacing', 'StudyInstanceUID',
'SeriesInstanceUID', 'SeriesNumber', 'AcquisitionNumber', 'InstanceNumber',
'ImagePositionPatient', 'ImageOrientationPatient', 'SamplesPerPixel', 'Rows',
'Columns', 'PixelSpacing', 'BitsAllocated', 'BitsStored', 'HighBit',
'PixelRepresentation', 'PixelData', 'shape', 'sampling'])
```

Q 3

You can also view a specific field/property of the metadata, by specifying it in square brackets. Show the *study date* of `brain_img`.

In [49]:
```
brain_img.meta['StudyDate']
#2010-02-04
```

Out[49]:
```
'20100204'
```

Q 4

The image you've loaded is a single 2D "slice" through the brain. What are the dimensions of the image, in voxels?

In [50]:
```
print('2d dimensions:', brain_img.shape)
```

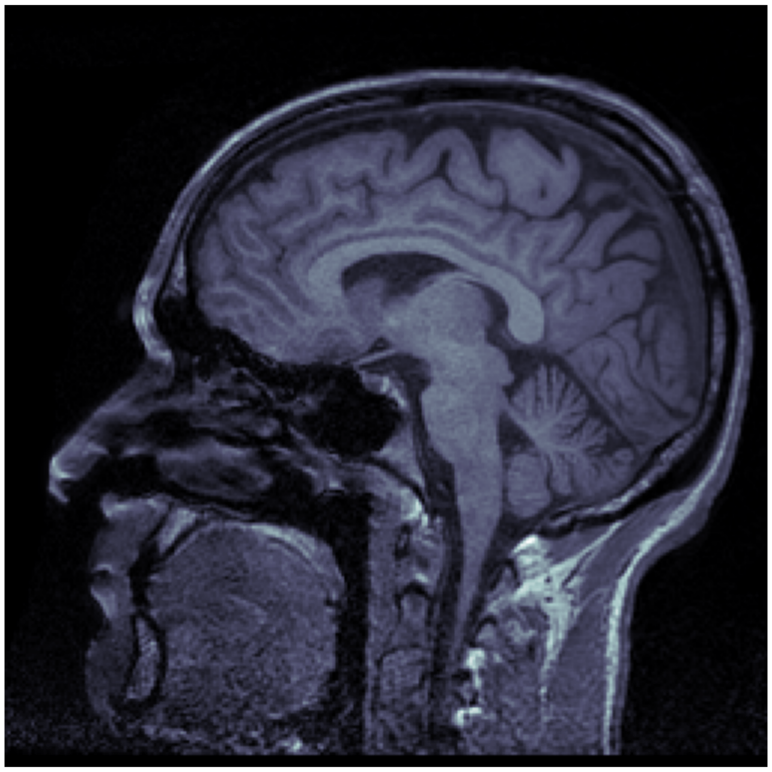
Out[50]:
```
2d dimensions: (256, 256)
```

Q5

Display the image. Use the `bone` colour map and turn off the axes.

In [6]:
```
fig, axes = plt.subplots()
axes.imshow(brain_img, cmap='bone')
axes.axis('off')
plt.show()
```

Out[6]:



Q6

A 3D Volume

Now we're going to load in a 3D volume, i.e., a whole series of slices that cover the entire brain. This is stored in a set of 184 DICOM files, each a slice through the brain. They are stored inside a folder called `SAG3DT1SPGR_4` inside the `data` folder. Use `iio.volread()` to load in this data set and assign it to `vol`. Note that if you pass a folder name to `.volread()`, it will read all the DICOM files in the folder.

In [7]:
```
vol = iio.volread("data/SAG3DT1SPGR_4", "DICOM")
```

Out[7]:
```
Reading DICOM (examining files): 1/184 files (0.5117/184 files (63.6%184/184 files
(100.0%)
  Found 1 correct series.
Reading DICOM (loading data): 184/184  (100.0%)
```

Q7

What are the dimensions of this image?

In [8]:
```
print('Shape of image array [Slices, Dimensions]:', vol.shape)
```

Out[8]:
```
Shape of image array [Slices, Dimensions]: (184, 256, 256)
```

The first value in the output above is the number of slices, and the last two values are the dimensions (number of voxels) within each slice.

Q 8

Visualize a slice

A 3D MRI image can be thought of as a collection of 2D slices. Most MRI scans, including this one, are collected slice-by-slice. In the present scan, the slices were acquired in the *axial* plane (axial slices are viewed as if you are looking down at the top of the head, or up from the feet as someone is lying in the scanner).

Because of how the data were acquired, slices are the first of the three dimensions in the image.

Below, plot the middle slice of the 3D MRI volume. To do this, you will need to pass not only the name of the image volume, but a slicing selector (here, "slicing" in the sense of lists or NumPy arrays, with an index value in square brackets) indicating the number of the slice you wish to view. Since slices are the first dimension of the array, you only need to pass the slice number; you don't need to specify the other dimensions.

This time, use the gray colourmap.

In [9]:
```
#Middles Image N = 184/2 [-1 due to 0 indexing] = 91

fig, axes = plt.subplots(nrows=1, ncols=1)
axes.imshow(vol[91, :, :], cmap='gray')
axes.axis('off')
plt.show()
```

Out[9]:

Q 9

View multiple slices

Since one slice is not very representative of the brain, we can use subplots to visualize a number of slices through the brain volume. Plot a figure with a 4 x 4 array of evenly-spaced slices as subplots.

Write a `for` loop to do this. A few hints:

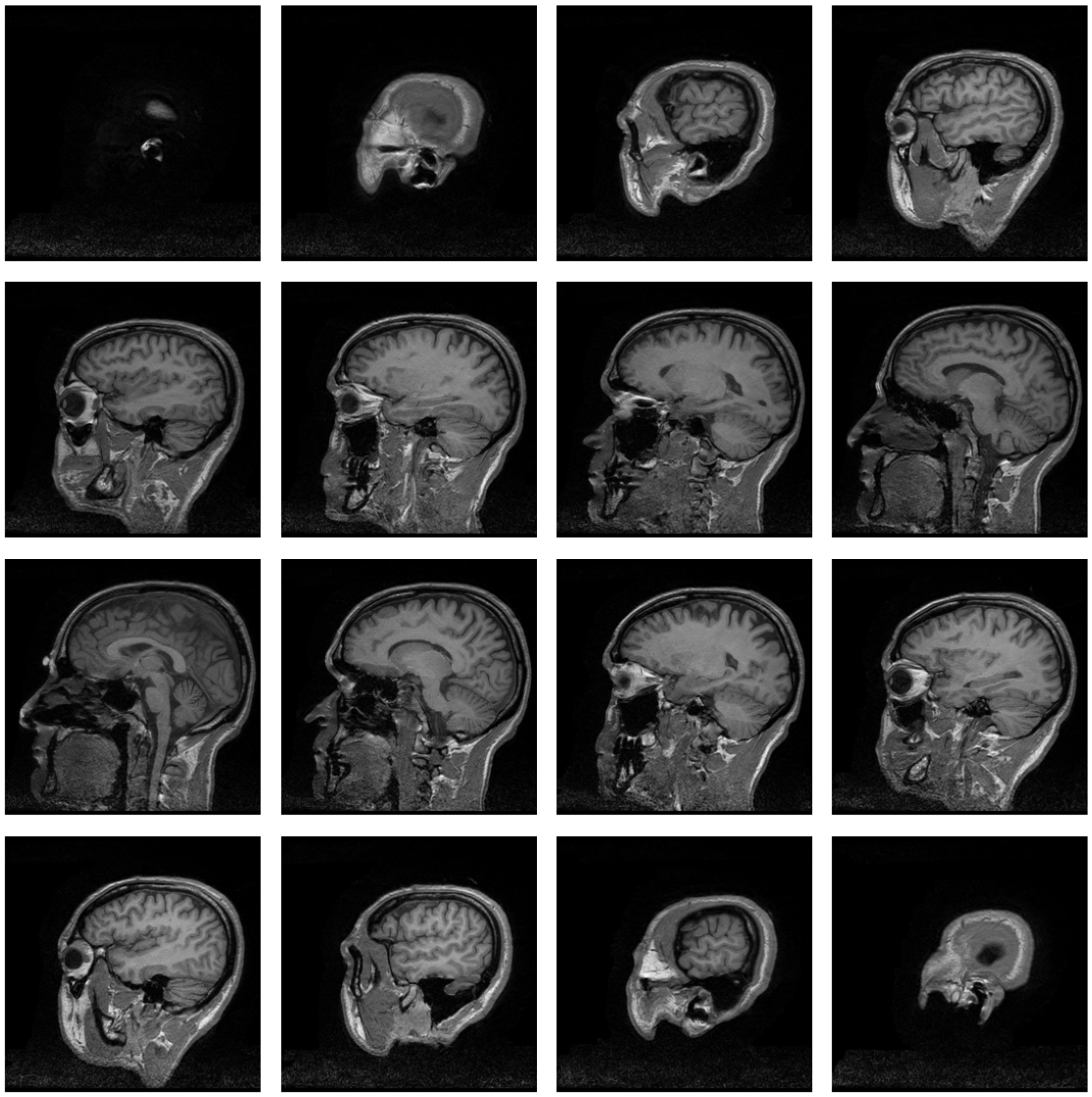
- unlike the DataCamp lesson and (probably) your plots above, here you should use object-oriented plotting. The `plt.subplots()` command is provided for you
- the `for` line should use the `range` function to go from 0 to the total number of slices (size of the slice dimension in the image), with the third `range` argument (the step) used to specify every 10th slice
    - Note that if you start at slice 0 and plot every 10th slice, using a range based on the total number of slices, you will get an error. This is because the number of slices (184) divided by 10 is more than the 16 subplots you have to work with. For this reason, you should specify a range that comprises only the middle 160 slices. This is what the `start` and `stop` variables we've defined are for
- the `for` loop should actually `enumerate()` the range function, returning two loop variables: the index of `enumerate`, and the image slice number from `range`
- use basically the same commands as in previous plots above to show the image, but use the object-oriented `ax[].imshow()` version rather than `plt.imshow()`
- you will use the enumerate loop variable to specify which subplot you are drawing each time through the loop. One issue with doing this (discussed in the lab session on Dec 1) is that for a figure with a 2D array of subplots, by default indexing is 2D (i.e., `[row, col]`), which makes it hard to refer to a specific subplot using the enumerate loop variable. The way to slove this is to use `axs.flat[]` rather than `axs[]` to refer to a

subplot. When you use `axs.flat[]`, it flattens the 2D array of axes to 1D, so you only to provide one integer inside the square brackets

- turn the axes off so no tick marks or numbers are shown

In [10]:
```
fig, axs = plt.subplots(4, 4, figsize=[8, 8])
start = int((vol.shape[0] - 160) / 2 )
stop = int(vol.shape[0] - start)
ax = axs.flatten()

for index, SliceNumber in enumerate(range(start, stop, 10)):
    ax[index].imshow(vol[SliceNumber], cmap='gray')
    ax[index].axis('off')


plt.tight_layout()
plt.show()
```

Out[10]:

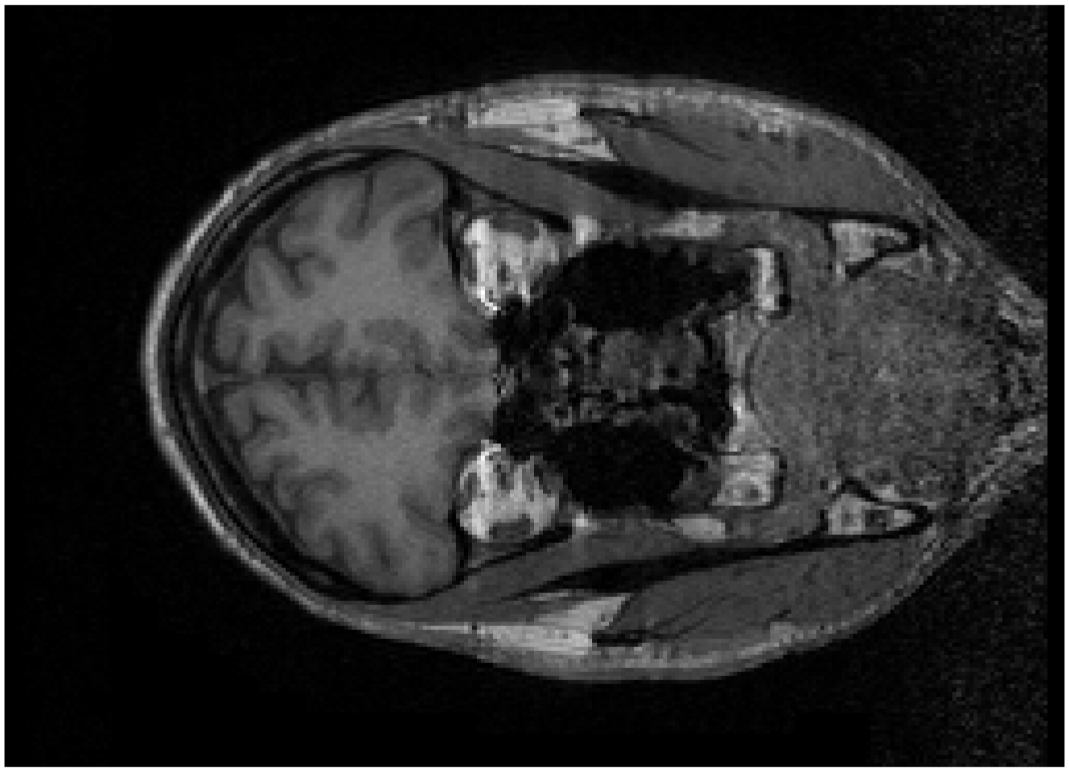


Q 10

Another cool thing we can do with a 3D image is visualize a slice in some other dimension. That is, although the image was acquired as 184 slices in the saggital plane, we can "reslice" the image to view it from a different perspective. Below, show the middle slice of the image in the coronal plane (the coronal plane would be as if you were viewing the person from the front). To do this, specify a number in the third dimension, with the other two dimensions shown as `:`. The orientation of the image will not be as expected; that's OK!

In [11]:
```
#Coronal Plane

fig, axes = plt.subplots(nrows=1, ncols=1)
axes.imshow(vol[ :, :,91], cmap='gray')
axes.axis('off')
plt.show()
```

Out[11]:

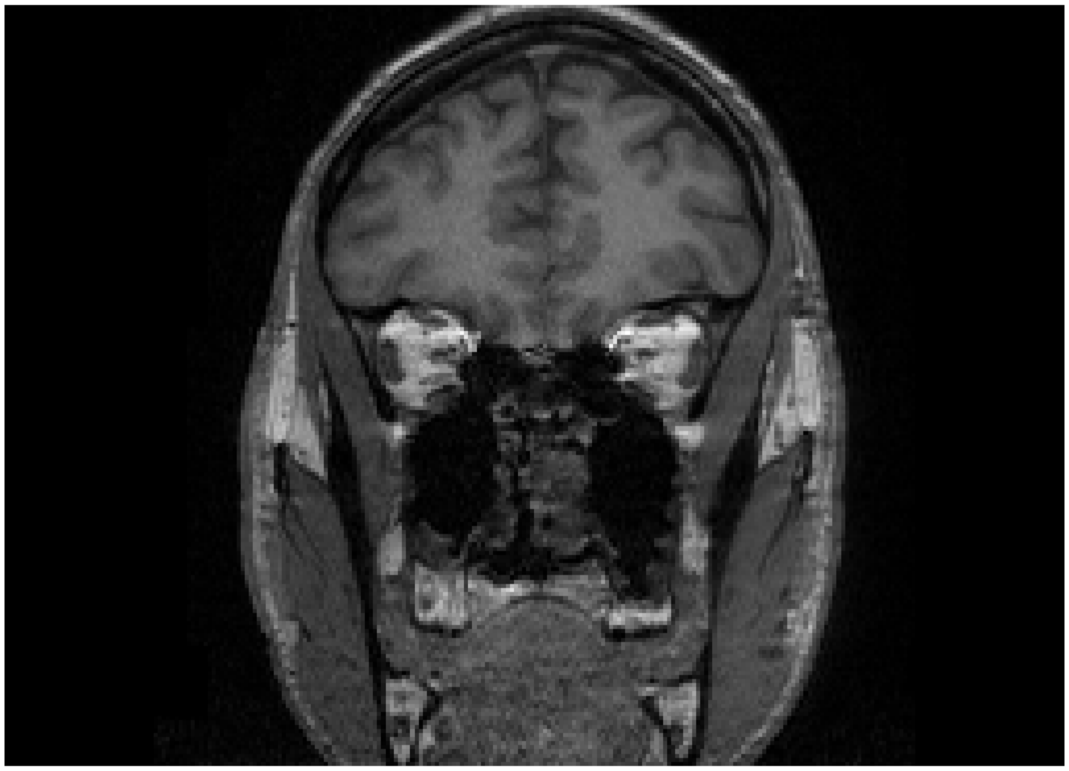


Q 11

We can turn the image right side up for the plot, using the `ndi.rotate()` function. This is described in chapter 4 of the DataCamp lesson. Copy and paste your code from the previous cell below, and embed the data specification (your slice of `vol`) inside the `ndi.rotate()` command.

In [12]:
```
#Coronal Plane rotated clockwise 90deg
vol = ndi.rotate(vol, angle=-90, reshape=False)
fig, axes = plt.subplots(nrows=1, ncols=1)
axes.imshow(vol[ :, :,91], cmap='gray')
axes.axis('off')
plt.show()
```

Out[12]:



Part B: Working with fMRI Statistical Maps

Now we'll see how we can apply these approaches to fMRI data. We're going to load in a statistical map — the results of an analysis across a group of individuals who performed a simple language production task in the MRI scanner. On each trial, a picture was presented and participants were instructed to say an action that would be appropriate for the object shown (e.g., if they saw a ball, they could say "throw" or "kick"). This verb generation task is a classic fMRI paradigm that does a good job of activating the language production network in the brain, including Broca's area (the left inferior frontal gyrus). Each fMRI scan lasted 5 minutes, and the study employed a block design, in which 30 sec periods of the verb generation task alternated with 30 s "control" periods when participants saw nonsense, scrambled images and were instructed to remain silent. This condition controlled for visual activation associated with seeing the pictures in the speech task, helping to ensure that any activation differences between the speech and control tasks were due to speech production and not visual activation.

The image that we load below is called a **statistical parametric map**. It's a 3D volume, like other MRI images we've worked with, but here the intensity value of each voxel is a statistical *z* score representing the results of a statistical test (*z* test) comparing the speech and control conditions, across the group of participants.

z scores are *standardized*, meaning that a particular *z* value is always associated with a particular *p* value. For example, *z* = 2.33 equates to a *p* = 0.01.

As a side note, these are in a different file format. The above images were in DICOM, but these are in NIfTI format. *NIfTI* stands for "neuroimaging informatics technology initiative", and was sponsored by the the National Institute of Mental Health and the National Institute of Neurological Disorders and Stroke with the aim of providing a standard file format that could be used across different software packages and labs working in neuroimaging research.

Q 12

Read fMRI data

SciPy's `imageio` package does not read NIfTI files, so we need to import and use a different package, NiBabel, which was written specifically for reading a variety of neuroimaging data formats (in spite of NIfTI's being an effort to standardize file formats, there are still a number of other formats in fairly widespread use). We imported NiBabel at the top of this file as `nib`. Here the function you need to use is `nib.load()`, and the file name is `language_zstat1.nii.gz` in the `data` folder.

In [13]:
```
# data/x
fmri_zstat = nib.load('data/language_zstat1.nii.gz')
```

Q 13

Print the type of `fmri_zstat`

In [14]:
```
type(fmri_zstat)
```

Out[14]:
```
nibabel.nifti1.Nifti1Image
```

The important thing to note from the output is that `fmri_zstat` is stored as a Python *object*, of a class provided by nibable. So it's a complex data structure, not just the data comprising a brain image.

Q 14

One important component of the NIfTI image is its `.header` property, which contains metadata for the image (similar to the `.meta` property when using `iio.imread`). Print the header of `fmri_zstat`:

In [15]:
```
print(fmri_zstat.header)
```

Out[15]:
```
<class 'nibabel.nifti1.Nifti1Header'> object, endian='<'
sizeof_hdr      : 348
data_type       : b''
db_name         : b''
extents         : 0
session_error   : 0
regular         : b'r'
dim_info        : 0
dim             : [  3  91 109  91   1   1   1   1]
intent_p1       : 0.0
intent_p2       : 0.0
intent_p3       : 0.0
intent_code     : z score
datatype        : float32
bitpix          : 32
slice_start     : 0
pixdim          : [-1.  2.  2.  2.  1.  0.  0.  0.]
vox_offset      : 0.0
scl_slope       : nan
scl_inter       : nan
slice_end       : 0
slice_code      : unknown
xyzt_units      : 10
cal_max         : 0.0
cal_min         : 0.0
slice_duration  : 0.0
toffset         : 0.0
glmax           : 0
glmin           : 0
descrip         : b'FSL5.0'
aux_file        : b''
qform_code      : mni
```

```
sform_code      : mni
quatern_b       : 0.0
quatern_c       : 1.0
quatern_d       : 0.0
qoffset_x       : 90.0
qoffset_y       : -126.0
qoffset_z       : -72.0
srow_x          : [-2.   0.   0. 90.]
srow_y          : [   0.     2.     0. -126.]
srow_z          : [  0.    0.    2. -72.]
intent_name     : b''
magic           : b'n+1'
```

The header is quite complicated! But some of the most important information is relatively easy to see above. For example, the `dim` field tells us the dimensions of the MRI image in the order *ndim*, *x*, *y*, *z* (the first value is the number of dimensions in the image; conceivably image files might have a fourth dimension if it was a time series). Likewise, the `pixdim` field shows the dimensions of the voxels in millimetres: they are 2 x 2 x 2 mm in size (the first value in this property, `-1`, is ignored). There's other information in the header, of course, but these are useful ones to know.

Q 15

Access and Visualize the Data

Since we will want to work with the data itself, we can extract that from the nibabel object and store it as a NumPy array, using the `.get_fdata()` method. Do this below, and assign the result to `fmri_zstat_data`. The output of the last line should confirm your result is a NumPy array

In [16]:
```
fmri_zstat_data = fmri_zstat.get_fdata()

type(fmri_zstat_data)
```

Out[16]:
```
numpy.ndarray
```

Q 16

What is the shape of `fmri_zstat_fdata`?

In [17]:
```
fmri_zstat_data.shape
```

Out[17]:
```
(91, 109, 91)
```

Q 17

We can get a sense of the range of values in the data by finding the min and max values. The image contains positive *z* scores, which represent stronger activation in the verb generation than control condition, as well as negative values, which represent stronger activation in the control condition. The latter are not of particular interest to us, but such "deactivations" commonly occur in just about every fMRI study. There is a whole field of research working to understand why this occurs and what it means; the set of brain areas that tends to be more active during rest conditions when people are "doing nothing" is referred to as the *default mode network*.

In the cell below, print the maximum and minimum data values of `fmri_ztat_data`:

In [18]:
```
print('Max= ' + str(fmri_zstat_data.max()))
print('Min= ' + str(fmri_zstat_data.min()))
```

Out[18]:
```
Max= 7.528600215911865
Min= -4.496517658233643
```

Q 18

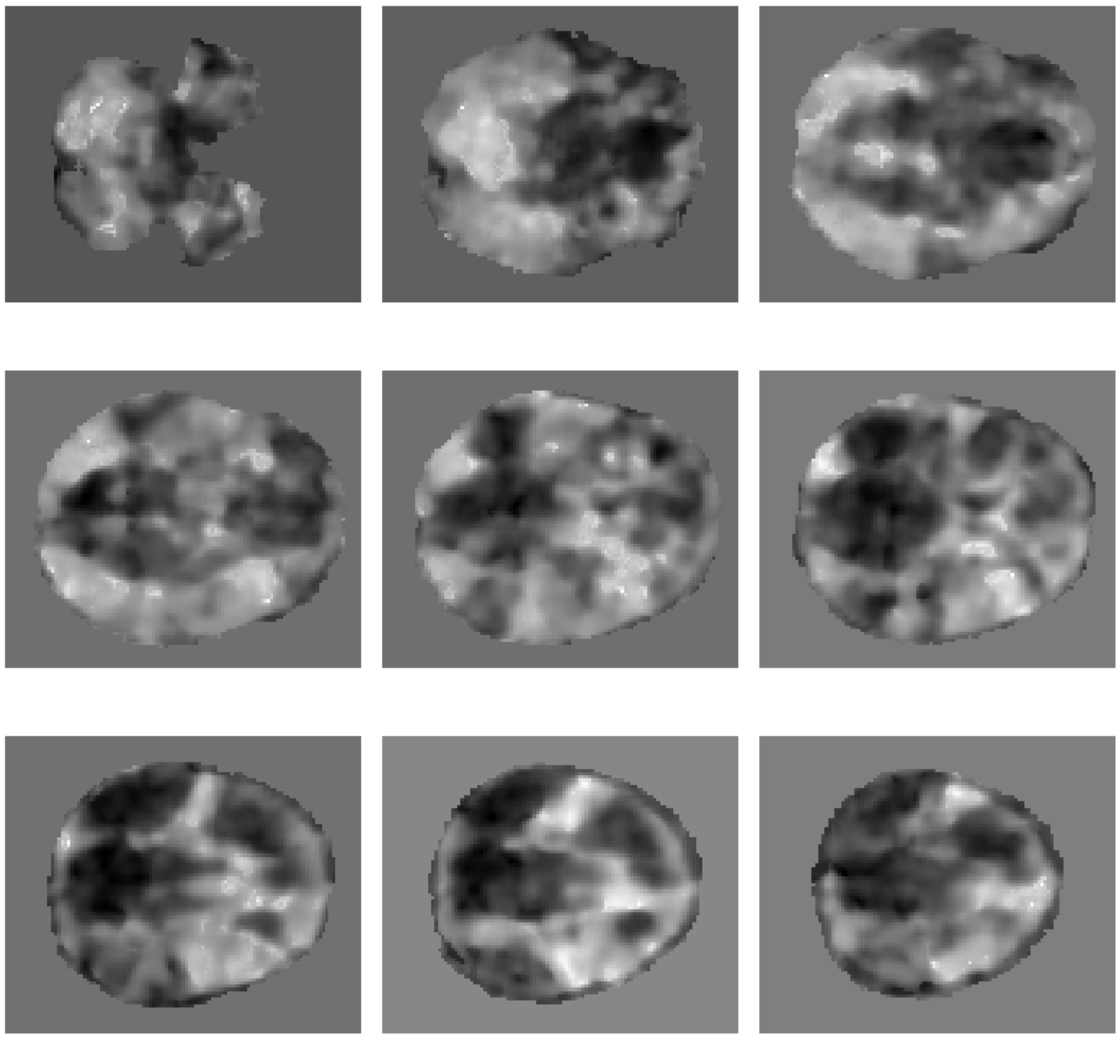
Plot a series of axial slices through this image. You can adapt the code you used above for this purpose. Here, we've provided you with the start, stop, and step values for the `range()` command you'll use in your `for` statement, because the top and bottom of the image volume don't actually contain any brain voxels.

Note that in this image, the axes of the image array are in the order *x*, *y*, *z*, so to get axial slices you'll need to slice along the *z* dimension.

You will see that although the shape of the image still looks brain-like, the intensity values make it look more "mottled" (blobs of black and white and grey). This reflects the fact that the brain is organized into distinct functional regions, that responded differently to the experimental manipulation.

In [19]:
```
fig, axs = plt.subplots(3, 3, figsize=[8, 8])
start = 20
stop  = 61
step = 5


for i, img in enumerate(range(start, stop, step)):
    axs.flat[i].imshow(fmri_zstat_data[:,:,img], cmap='gray')
    axs.flat[i].axis('off')

plt.tight_layout()
plt.show()
```

Out[19]:



Q 19

This is a case where a different colour map can help with visualization. Since the data have positive and negative values, with 0 meaning "no activation difference between conditions", it makes sense to use a colour map that has different colours for positive and negative, and no colour close to zero. The `seismic` colour map in Matplotlib is designed for exactly this purpose, using a gradient of reds for positive, and blues for negative, values.

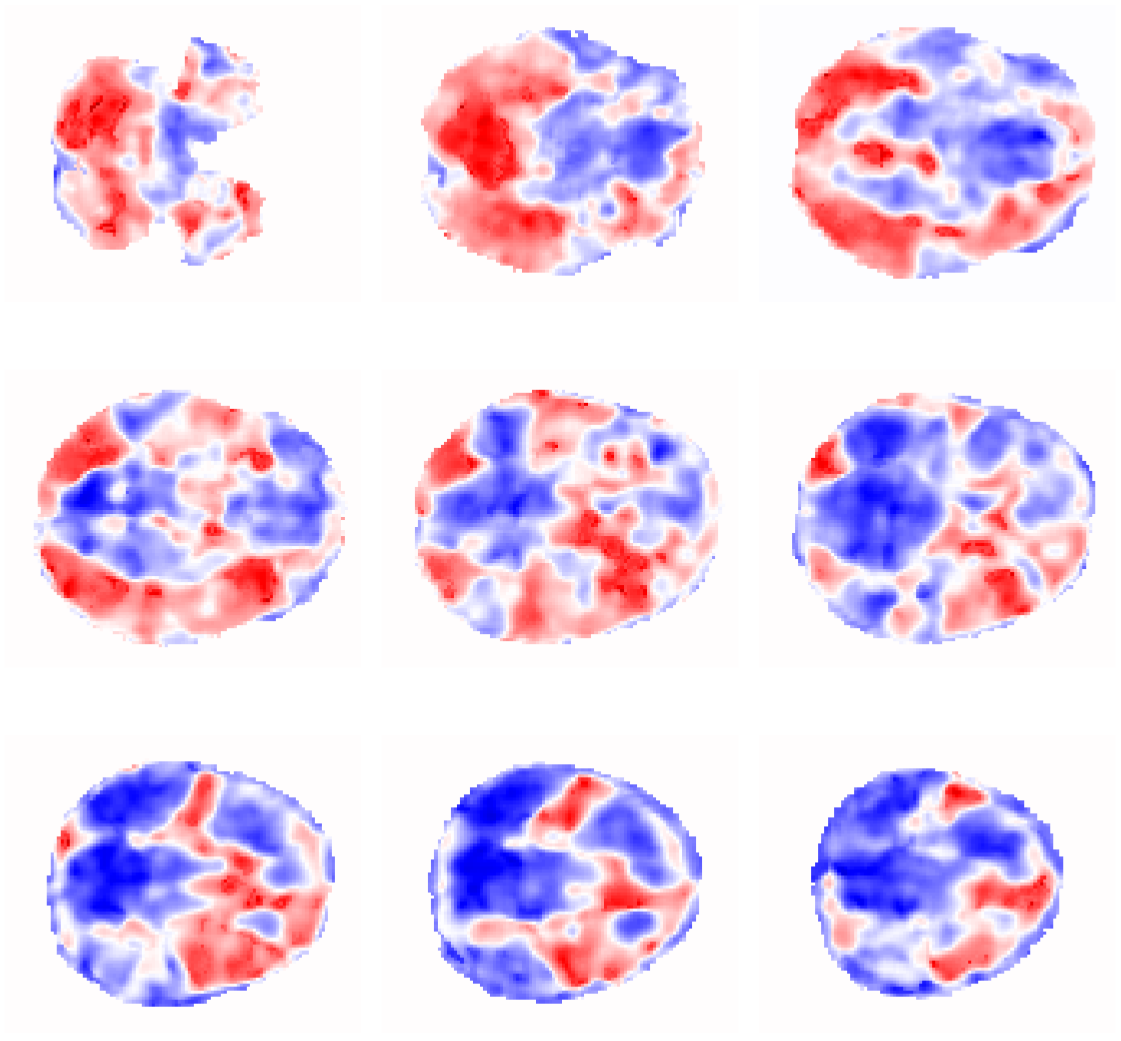
We also provide the arguments `vmin` and `vmax` to the plot command, to set the upper and lower ranges that the colours map to. This is important to ensure that all of the panels are scaled the same; you'll notice above that when we don't do this, the background (non-brain areas) intensity varies in each plot, suggesting that something is not right about how we're plotting the data.

In the cell below, copy and paste your code from the previous cell to again plot 9 slices through the image. Make the following changes to the code:

- use the `seismic` colour map in`imshow()'
- also as arguments to `imshow()`, use `vmin=` and `vmax=`. As the max value, use the max *z* score you found above. As the negative value, use the *negative* of the max *z* score (not the min *z* score! It's important that the min and max are the same, so that the colourmap sets white to zero)

In [31]:
```
fig, axs = plt.subplots(3, 3, figsize=[8, 8])

start = 20
stop  = 61
step = 5

for i, img in enumerate(range(start, stop, step)):
    axs.flat[i].imshow(fmri_zstat_data[:,:,img], cmap = 'seismic',vmin=
-7.5,vmax=7.5)
    axs.flat[i].axis('off')

plt.tight_layout()
plt.show()
```

Out[31]:



Q 20

How many voxels are in the image? This is the multiple of the three dimensions of the image array

In [46]:
```
# Compute volume
nonzero_voxel_count = np.count_nonzero(fmri_zstat.get_fdata())
voxel_volume = np.prod(voxel_dims)
nonzero_voxel_volume = nonzero_voxel_count * voxel_volume

print("Number of non-zero voxels = {}".format(nonzero_voxel_count))
```

Out[46]:
```
Number of non-zero voxels = 264400
```

Q 21

False positive rate

Since our objective is to identify Broca's area and any other brain area that is more active during speech production, we will next *threshold* the image to only show the *z* scores above our statistical threshold.

Statistical thresholding in fMRI is actually a big topic, because it's more complex than in other areas of psychology or neuroscience that you may have encountered. We're performing a statistical test at *each* voxel — so the number of statistical tests is the value you obtained in answering the previous question. So a conventional *p* threshold of .05 — which means that 5% of "significant" findings would occur by chance — would result in over thousands of false positives just by chance.

In the cell below, show the number of false positives you would obtain, based on the number of voxels in the image this would be 5% of the number of voxels in the image.

```
In [51]: print('5% of Number of non-zero Voxels:',nonzero_voxel_count * 0.05 )
Out[51]: 5% of Number of non-zero Voxels: 13220.0
```

Thresholding the statistical image

In "real" fMRI work we use sophisticated routines to correct for multiple comparisons. Here, for simplicity we'll just apply a threshold of *p* < .0001. This is more conservative (fewer false positives) than *p* < .05, though not as accurate as what would be used in real fMRI research.

`scipy.stats` provides a convenient function to determine the *Z* value corresponding to any *p* value. Note that we need to use the inverse of our desired *p* threshold (1 - *p*). This is provided for you, so just run the cell below and move on.

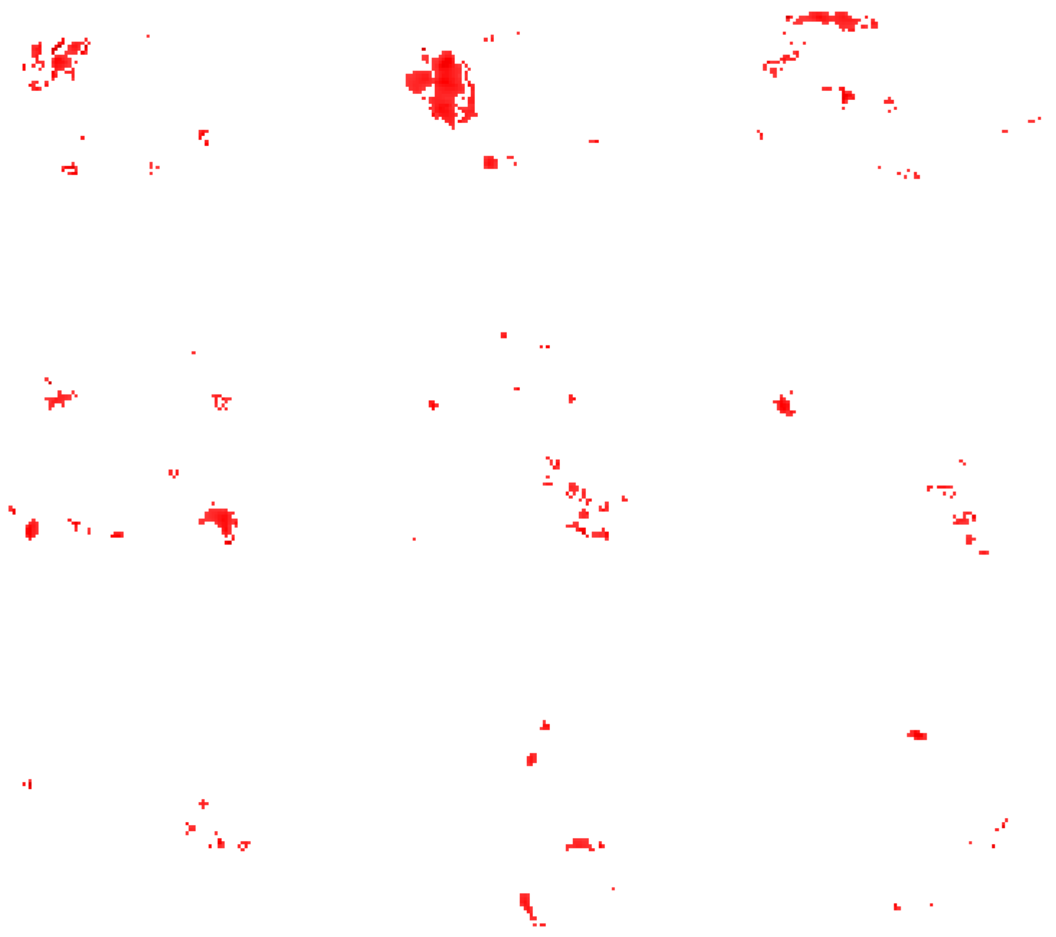
```
In [52]: z_thresh = scipy.stats.norm.ppf(1 - .001)
         print(z_thresh)

Out[52]: 3.090232306167813
```

Q 22

Copy and paste your code from above that showed 9 slices through the *z* map with the seismic colour map, and this time plot the image thresholded to show only voxels with values above `zthresh`. To do this, you will first need to make a mask using `np.where()`. Assign the mask to `thresh_zstat`. Voxels above threshold in `thresh_zstat` should take the values of `fmri_zstat_data`, and values below threshold should be set to `np.nan`.

Don't worry if you don't see a lot! You should see some bits of red, but a lot less than in the previous image.

```
In [62]: fig, axs = plt.subplots(3, 3, figsize=[8, 8])

         start = 20
         stop  = 61
         step = 5

         # YOUR CODE HERE
         thresh_zstat = np.where( fmri_zstat_data > z_thresh, fmri_zstat_data, np.nan)
         for i, img in enumerate(range(start, stop, step)):
             axs.flat[i].imshow(thresh_zstat[:,:,img], cmap = 'seismic',vmin= -7.5,vmax=7.5)
             axs.flat[i].axis('off')


         plt.tight_layout()
         plt.show()

Out[62]:
```



Q 23

Of course, those "blobs" don't tell us much without being able to visualize them relative to the brain anatomy. To do this, we can overlay the thresholded *z* map on a greyscale brain image. The easiest one to use is a "raw" fMRI image from the same scan, because this has the same resolution and orientation as our statistical map. There are ways to use a higher-quality anatomical scan, but we won't add that complexity to our code here.

The anatomical underlay for this data set is the file `bg_image.nii.gz` in the `data` folder. Load it, and assign its data to a variable called `underlay`.

In [57]:
```
underlay = nib.load('data/bg_image.nii.gz')
underlayf = underlay.get_fdata()
```

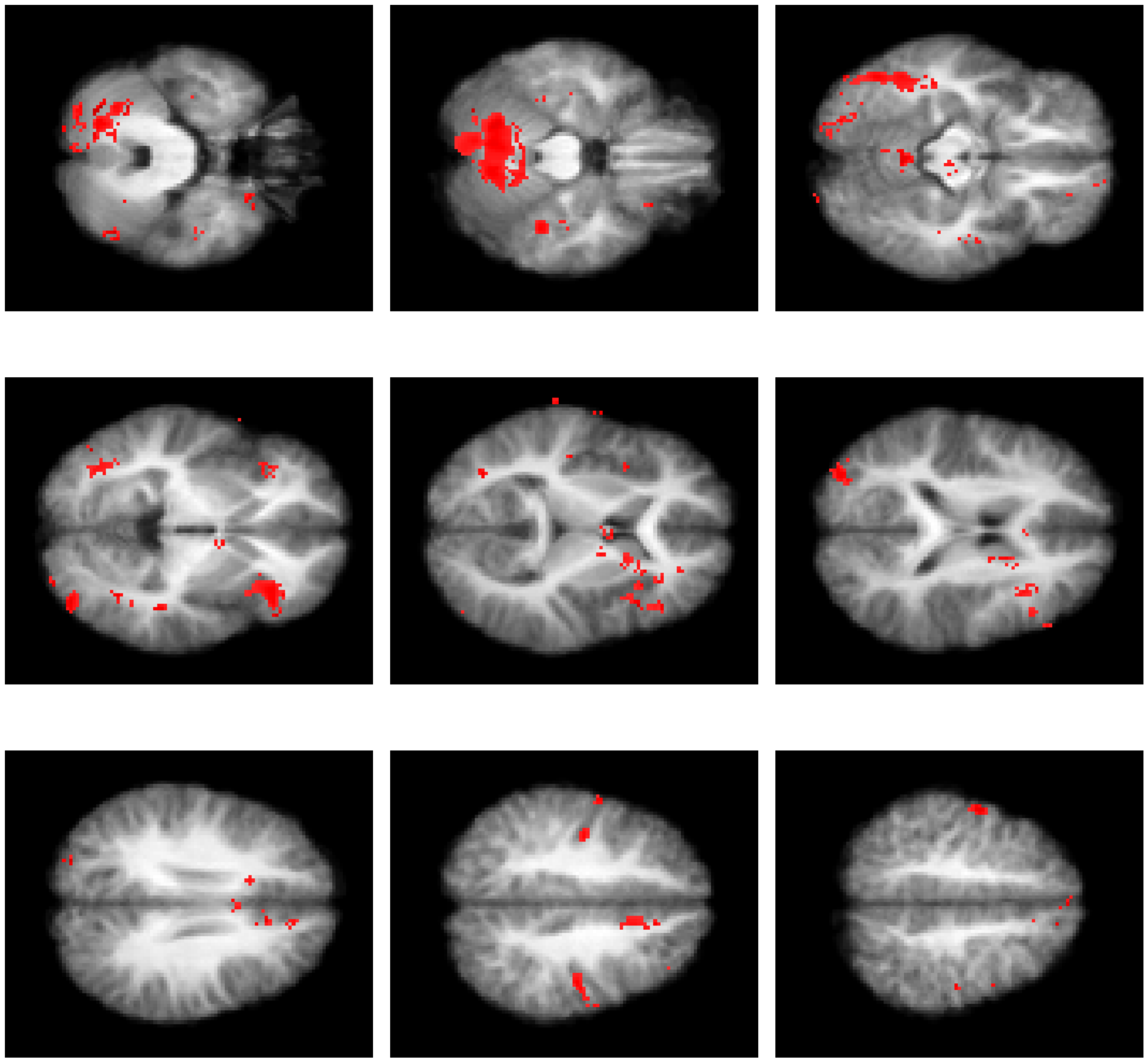
Q 24

Now, copy and paste the code from above that plotted `thresh_zstat`. The only change you need to make is to add one line of code right *above* the `.imshow()` line that plots `thresh_zstat`, but the image it plots should be `underlay`, and the colourmap should be gray (don't add any other arguments). Be sure that you're using the same slicing code on the data in both `plt.imshow()` commands, to ensure you're plotting the same slice for the overlay and underlay.

In [64]:
```
fig, axs = plt.subplots(3, 3, figsize=[10, 10])

start = 20
```

```
stop  = 61
step = 5

thresh_zstat = np.where( fmri_zstat_data > z_thresh, fmri_zstat_data, np.nan)
for i, img in enumerate(range(start, stop, step)):
    axs.flat[i].imshow(underlayf[:,:,img], cmap='gray')
    axs.flat[i].imshow(thresh_zstat[:,:,img], cmap = 'seismic',vmin= -7.5,vmax=7.5)
    axs.flat[i].axis('off')

plt.tight_layout()
plt.show()
```

Out[64]:



If you did things right, you should see particular activation in the left hemisphere Broca's area in the middle row, left-most column, along with several other areas, including fairly extensive activation in the cerebellum.

Note that the front of the head is pointing right in these images, and the left side of the head is on the *bottom* of each image. This is due to a convention in radiology, in which the head is typically plotted as if you were standing outside the MRI scanner at the participant's feet, looking up at the bottom of the head. This is, understandably, a huge source of confusion in neuroimaging research! Most researchers flip their images so that the left side of the head is on the left side of the image. It's also good practice, when publishing neuroimaging figures like these, to add left/right labels to figures, or note this in the figure caption. No need to worry about doing this here, this is just an explanation.

In [0]: