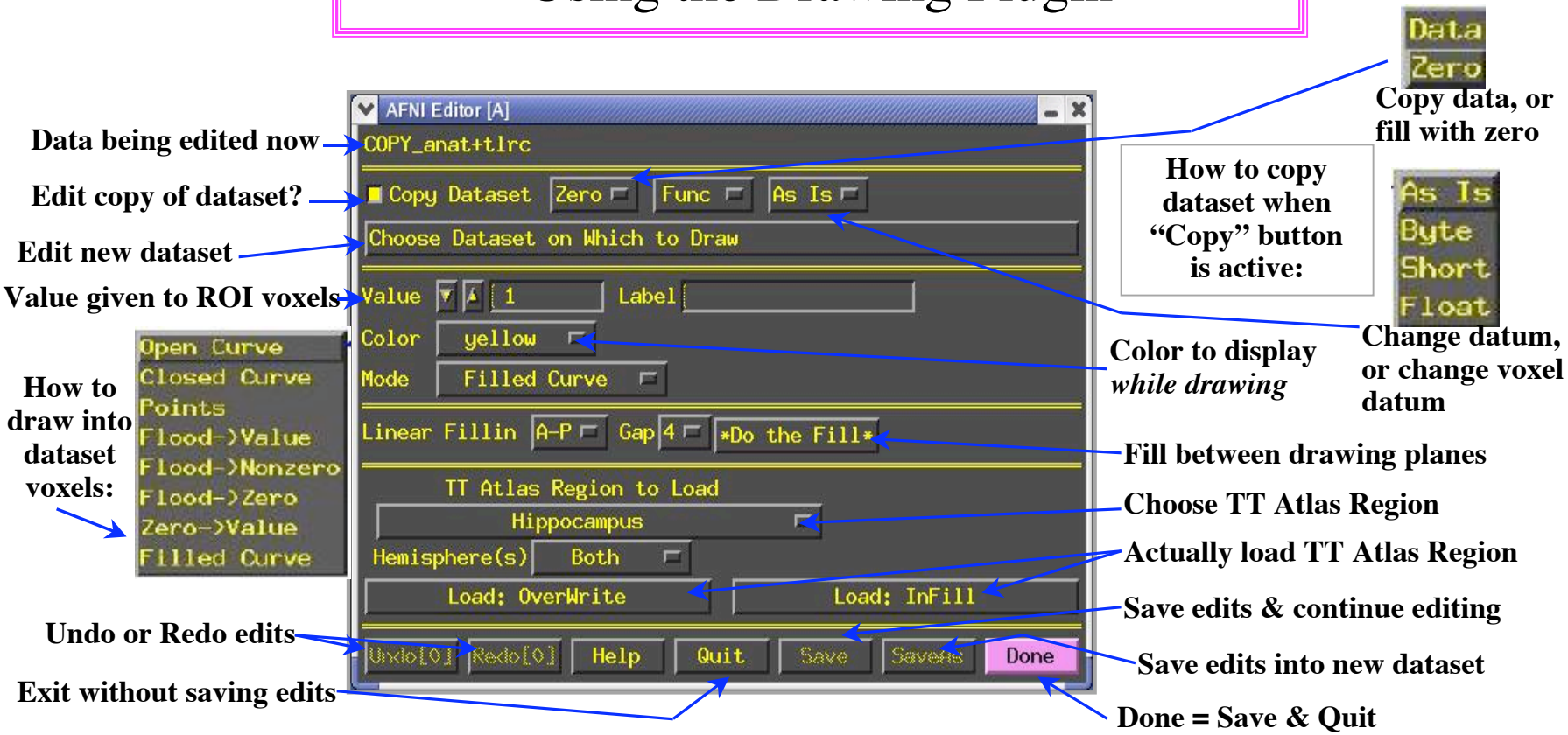
Region of Interest Drawing and Usage in AFNI

- Goal: Manually select Regions of Interest (ROIs) based on anatomical structures, then analyze functional datasets within these regions
  - ★ E.g., Someone doing a study on emotion may want to draw an ROI mask that includes the amygdala only in order to analyze the voxels that fall within that brain region
  - ★ This method relies on 'a priori' assumptions about localization of brain function
- Alternative procedure for selection of ROIs: Analyze functional dataset for *entire* brain first, then focus on geometrically connected clusters of 'activity' (supra-threshold voxels in some functional statistical map)
  - ★ i.e., analyze the entire brain first and *then* pinpoint interesting areas of activity and do further analyses on those areas (**`3dclust`** or **`3dmerge`** can assist you in finding those larger "blobs" or clusters of activity)
    - ➥ Even with this method, you might want to manually adjust the ROIs
- ROIs are stored as regular AFNI datasets; it is only the user (you) that decides whether a particular dataset is a ROI mask
  - ★ Nonzero voxels are "in" the mask; zero voxels are "outside"

- **Quick outline of procedure:**
  1. On the main AFNI control panel, set the anatomical underlay dataset (with [Switch UnderLay]) to be what you want to draw on -- usually a SPGR or MP-RAGE type of dataset
     - i.e., the anatomical underlay will serve as our guide or template for drawing the ROI mask
  2. Start the **Draw Dataset** plugin (this is our ROI-creating plugin):
     - [Define Datamode] → [Plugins] → [Draw Dataset]
  3. Create an *all zero* anatomical overlay dataset with the **Draw Dataset** plugin. This is the beginning of our ROI mask. At this point, the anatomical overlay is empty - i.e., all the voxel values are zero
     - This anatomical overlay *must* have the same geometry as the anatomical underlay, i.e., it has the same voxel size as the underlay, the same xyz-grid spacing, etc…, since drawing/editing is done on a voxel-by-voxel basis
       - Think of the anat overlay as a blank piece of tracing paper that is the same size as the template underneath. The blank overlay will be used to trace portions of the underlay. Voxels *inside* the traced portion will make up the ROI mask. Values *outside* the traced region are irrelevant (zeros)
     - Note: You could also edit an already-existing ROI mask (that you created earlier) at this step

4. To view and begin drawing an ROI mask (or several ROIs) on this blank anatomical overlay dataset, go to the main AFNI interface and **[Switch Overlay]** to be the empty anatomical dataset you will be editing. Also turn the overlay ON with **[See OverLay]**
5. Start drawing the ROI mask into this blank anatomical overlay dataset. Voxels inside the ROI mask will receive a non-zero value (you decide what value to give them). Values outside the ROI mask will remain zero
   - ➥ Be sure to save the results by pressing **[Save]**, **[SaveAs]** or **[DONE]** in the ROI plugin GUI (**[Quit]** will exit the ROI plugin without saving your work)
6. Convert the anatomical-resolution ROI dataset into a dataset at the resolution of the functional (statistical) datasets you want to analyze with the ROI
   - ➥ Note: The ROI and functional datasets may already be at the same resolution, if you are operating in `+tlrc` coordinates
   - ➥ Resolution conversion of masks is done with program **3dfractionize**
7. Use programs **3dmaskave**, **3dmaskdump**, and **3dROIstats** to extract ROI-based information about functional datasets
   - ➥ Also can use the **ROI Average** plugin to extract interactively the average of a dataset over a ROI

Using the Drawing Plugin



- **Critical things to remember:**
  - ★ You should have [See OverLay] turned on, and be viewing the same overlay dataset in AFNI as you are editing
    - ➥ Otherwise, you won't see anything when you edit!
  - ★ When drawing, you are putting numbers into a dataset brick
    - ➥ These numbers are written to disk only when you do [Save], [SaveAs] or [Done]; before then, you can [Quit] (or exit AFNI) to get the unedited dataset back

- **Step 1: Load a dataset to be edited (for ROI creation):**
  - ★ **[Choose Dataset]** button gives you a list of datasets that
    (a) Actually have brick data with only one sub-brick;
    (b) Are at the same voxel dimension, grid size, etc., as current anat underlay
  - ★ When you are starting, you probably don't want to edit an existing dataset -- i.e., you don't want to write on the underlay itself; you just want to use it as a template and draw on a blank overlay that shares the same geometry as that existing underlay dataset
    - ➥ To do this, you must create an all-zero *copy* of the anatomical underlay (by "copy" we mean the all-zero dataset shares the same geometry as the underlay, not the same voxel data values)
    - ➥ To create an all-zero copy, click the **[Copy]** button on (from the **Draw Dataset** plugin GUI) and set the controls to its right to **[Zero]** and **[As Is]** (the **[Func]** button is a relic of the program and is now obsolete -- ignore)
      - ➪ **[Data]** would make a copy of the underlay dataset with the actual voxel data values. **[Zero]** copies the geometry of underlay, but gives each voxel a data value of zero (this latter option is usually want you want when starting out)

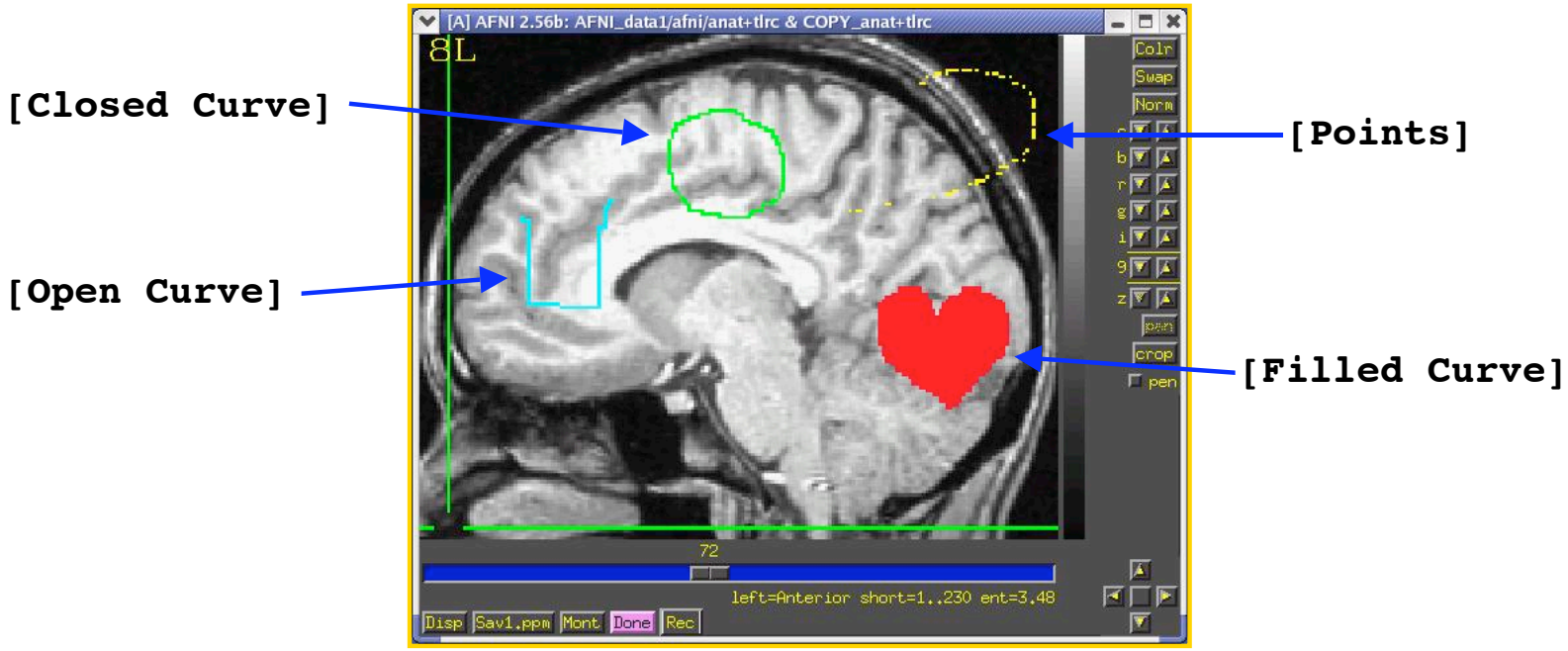
- [As Is] keeps the voxel values in the copy as the same type as in the original underlay; you can also change the voxel values to be stored as:
  - [Byte] (integer values: 0..255) ⇒ 1 byte each
  - [Short] (integer values: -32767…32767) ⇒ 2 bytes each
  - [Float] (fractional values) ⇒ 4 bytes each
  - Bytes and Shorts make the most sense for ROI masks, where you are essentially attaching labels to voxels
- Click on [Choose Dataset], select the dataset you want a copy of (e.g., the anatomical underlay), and then press [Set]

- **Step 2: Drawing the ROI (or ROIs):**
  - *Choose the value to draw into the anatomical overlay dataset* (recall that all values in the copied/blank overlay dataset are zero at this point)
    - If you drawing only one ROI, then the default value of 1 is good
    - If you are drawing multiple ROIs, then you should choose a different numerical value for each so that they can be distinguished later
      - Pencil and paper are our friends -- write down which number corresponds with which ROI for later recall!
  - *Choose the drawing color*
    - This is the color that is shown *while you are drawing*
    - Color choice is only important to give a good contrast with underlay, so don't obsess too much over this

- After you finish a drawing motion, the voxels you drew will be filled with the drawing value, the image will be redisplayed, and the colors will be determined by the **[Define OverLay]** control panel

★ *Choose the drawing mode*

- **[Filled Curve]**
  Drawing action produces a continuous closed-ended curve (default setting)
- **[Open Curve]**
  Drawing action produces a continuous open-ended curve
- **[Closed Curve]**
  Drawing action produces a continuous closed-ended curce
- **[Points]**
  Only points actually drawn over are filled (used to “touch up” and ROI)



➥ [Flood→Value]

Flood fills space outward from the drawing point, stopping when the flood hits the current drawing value (used to fill a closed curve)

➥ [Flood Nonzero]

Drawing action produces a continuous closed-ended curve (filled inside)

➥ [Zero→Value]

Floods voxels with zero until the flood hits nonzero voxels (you can also do this more easily with Filled Curve, value=0)

➥ [Flood→Nonzero]

Flood fills outwards from drawn point, stopping when the flood hits any nonzero voxel (used to fill between regions):

- ★ *Actually draw something*
  - ➥ Drawing is done with mouse Button 2 (“middle” button) in 2D slice image
  - ➥ Hold the button down in the image window during a single drawing action
  - ➥ While the drawing action is happening, the chosen drawing color will trace the screen pixels you draw over
  - ➥ When you release the button, these pixels are converted to voxels, and the dataset is actually changed, using the drawing value and drawing mode you selected
  - ➥ At this point, the color of the drawn region will change to reflect the drawing value and the setup of the **[Define OverLay]** control panel
- ★ **[Undo]** button will let you take back the last drawing action (you can go “undo” many levels back, i.e., multiple undo function)
- ★ You can draw on one 2D slice image at a time
  - ➥ If you draw on a montage display, only screen pixels overlaying the first image you Button 2 click in will count
  - ➥ While drawing, if you cross over between sub-images in the montage, unexpected effects will result
    - ➪ But there is always **[Undo]** to the rescue!

- **Step 3: Save your results:**
  - ★ [Save] will write the current dataset values to disk (overwriting any existing .BRIK file, i.e., if you had edited this ROI earlier, the new changes would overwrite the old file)
    - ➥ You could also then choose another dataset to edit
  - ★ [Save As] will let you write the current dataset to disk under a new name, creating a new dataset, then continue editing the new dataset
  - ★ [Quit] exits editing and closes the plugin window, without saving to disk any changes since the last [Save]
    - ➥ Exiting AFNI has the same effect
  - ★ [Done] is equivalent to **[Save]** then **[Quit]**
- **Optional Drawing Steps:**
  - ★ [Linear Fillin] lets you draw a 3D ROI not in every slice, but in every third slice (say), and then go back and fill in the gaps
    - ➥ For example, if you draw in coronal slices, then you want to fill in the [A-P] direction (the default)
    - ➥ If you draw every nth slice, then you want to set the Gap to n-1
    - ➥ Line segments of voxels in the fillin direction that have a current drawing value at each end, and have no more than [Gap] zero voxels in between, will get their gap voxels filled with the drawing value
      - ➪ After you try this, you will probably have to touch up the dataset manually

- This operation can also be done with program **3dRowFillin**, which creates a new dataset

★ [TT Atlas Region to Load] lets you load regions from the Talairach Daemon database into the dataset voxels

- Requires that you be drawing in +tlrc coordinates, or at least have a transformation from +orig →+tlrc computed in the current directory
- Choose a region to draw into the dataset (e.g., Hippocampus)
- [Load: Overwrite] will fill all voxels in the region with the drawing value
- [Load: Infill] will fill only voxels in the region that are currently zero
  - You probably want to edit the results manually to fit the specific subject

• **Drawing and Rendering at the Same Time** (totally fun, and maybe useful)

★ You cannot draw into the rendering plugin, but you can use it to see in 3D what you are drawing in 2D

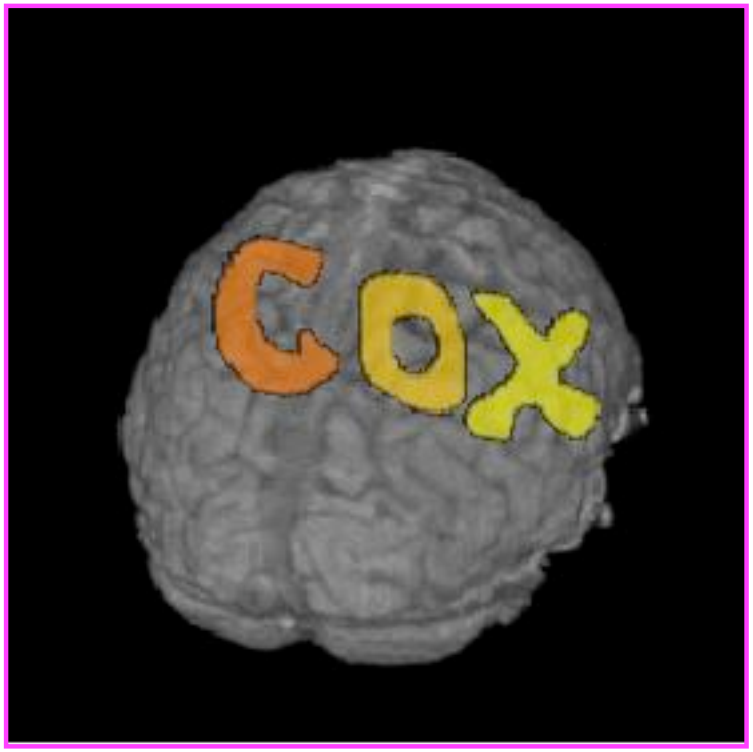
- If you meet the criteria for rendering (usually in +tlrc coordinates)

★ How to set up the renderer:

- Choose the underlay to be the current anatomical dataset (or a "scalped" version, from **3dIntracranial**)
- Choose the overlay dataset to be the dataset you are editing
- Turn on [See Overlay]
- Set [Color Opacity] to [ShowThru] (or **ST+Dcue**)

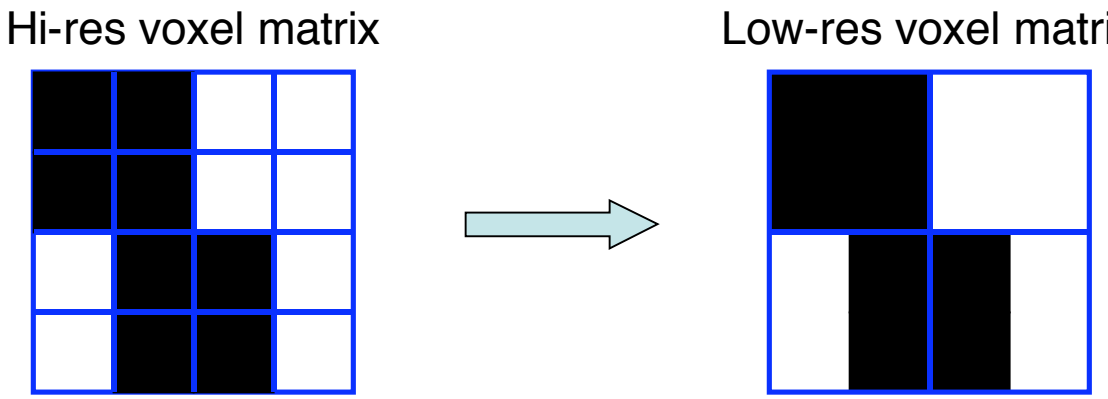
- Turn on [DynaDraw]
- Drawing in a 2D image window immediately triggers a redraw in the rendering window

  (if the 2D and 3D overlay datasets are the same)
- This is only useful if your computer is fast enough to render quickly (<1 sec per frame)



Things to Do with ROI Datasets

- ROIs are used on a voxel-by-voxel basis to select parts of datasets (usually functional datasets)
- If you draw at the anatomical resolution and want to use the ROI dataset at the functional resolution, you probably want to convert the high-resolution ROI dataset to a low-resolution dataset (unless you're working in `+tlrc` coordinates)
  - ★ E.g., hi-res anatomical ROI resampled to low-res functional dataset:



  - ★ Each voxel inside the ROI is given a nonzero value (e.g., 4; values outside the ROI are zeros. When the resolution is changed, what do you do with a voxel that's only partially filled by the ROI?

- **3dfractionize** does this resolution conversion:

```
3dfractionize  -template func+orig
               -input ROI_high_res+orig
               -clip 0.5 -preserve -prefix ROI_low_res
```

★ **-template func+orig** → The destination grid you want your ROI grid to be resampled to (we're going from high to low resolution here). Our output dataset **ROI_low_res+orig** will be written at the resolution of **func+orig**

★ **-input ROI_high_res+orig** → Defines the input high-resolution dataset (that needs to be converted from high resolution to low resolution)

★ **-clip 0.5** → Output voxels will only get a nonzero value if they are at least 50% filled by nonzero input voxels (you decide the percentage here). E.g., when going from high to low res, keep a label a voxel as part of the ROI if it is filled with at least 50% (or more) of the voxel value. For example:

This voxel is 80% filled with the ROI value -- keep it

This voxel is 30% filled with the ROI value -- lose it

★ **-preserve** → once it has been determined that the output voxel will be part of the ROI, preserve the original ROI value of that voxel (and not some fraction of that value). For example, if our ROI mask has values of "4":

This voxel is 80% filled with the ROI value -- keep it.

Without the `-preserve` option, this voxel would be given a value of "3.2" (i.e., 80% of "4").

With `-preserve`, it is labeled as "4"

- **`3dresample`** does this conversion as well:

```
3dresample -master func+orig
           -inset ROI_high_res+orig
           -rmode NN
           -prefix ROI_low_res
```

★ **`-master func+orig`**: the destination grid we want our ROI mask resampled to

★ **`-inset ROI_high_res+orig`**: The ROI mask dataset that is being resampled from high to low resolution

★ **`-prefix ROI_low_res`**: The output from **`3dresample`** -- a low res ROI mask that corresponds with the voxel resolution of our functional dataset

★ **`-rmode NN`**: If a voxel's "neighbor" is included in the ROI mask, include the voxel in question as well

- Note: A ROI is defined by the values stored in voxels of the 'mask' dataset
  - ★ Contiguity of voxels has no meaning to the ROI software described below
  - ★ Two voxels are in the same ROI if they have the same value in the mask dataset (i.e., it doesn't matter where they are located in the volume)

- **3dmaskave**
  - ★ Program to compute the average of voxels from a functional dataset, with voxels selected from a mask dataset (interactive version: **ROI Average** plugin)
  - ★ Example:

```
3dmaskave  -mask ROI_fred+tlrc  -mrange 1 4
          `func_fred+tlrc[2,5,7]`
```

Will print out (in the shell) one line of data for each of the 3 selected sub-bricks from the input dataset **func_fred+tlrc**:

```
44.287 [137 voxels]
27.021 [137 voxels]
-33.22 [137 voxels]
```

←sub-brick #2 values averaged across all 4 ROIs
←sub-brick #5 values averaged across all 4 ROIs
←sub-brick #7 values averaged across all 4 ROIs

➥ The output above represents the average of all the voxels collapsed across ROI's 1, 2, 3, 4, (specified by `-mrange 1 4`) in our mask dataset **ROI_fred+tlrc** for sub-bricks #2, #5, and #7 from dataset **func_fred+tlrc**

- **-mrange 1 4**: in this example, our mask dataset **ROI_fred+tlrc** has four ROIs draw within it. This option tells the program to take those 4 ROIs and compute an overall average (collapsed across ROIs) for each voxel that falls within the ROIs
  - The exact numbers to average is determined by the sub-bricks (#2, #5, #7) specified in our functional dataset, **fred+tlrc**

- Add the **-q** option to the above command line to suppress the voxel count printout (in this example, `-q` suppresses "`[137 voxels]`" from appearing in the shell
  - E.g., **44.287**
    27.021
    -33.22
  - This way you can copy and paste the single column of averages to a `1D` (i.e., text) file for later use in AFNI

- Instead of having the results of **3dmaskave** spewed into the shell, you can redirect (**>**)the results into a text file and save them for later use:

```
3dmaskave -mask ROI_fred+tlrc -mrange 1 4  -q
          `func_fred+tlrc[2,5,7]` > fred_func_ROI1-4.1D
```

- **3dmaskdump**
  - ★ Program that dumps out all voxel values in a dataset that fall within the ROI of a given mask dataset
  - ★ Example:

```
3dmaskdump -noijk -mask ROI_fred+tlrc
           -mrange 1 1 `func_fred+tlrc[2,5,7]`
```

➥ The output appears in the shell (unless you redirect it (>) into a text file) and shows 3 columns of numbers, representing the voxel values for functional sub-bricks 2, 5, and 7, that fall within the ROI mask:

voxel 1	12	37	42
voxel 2	27	0	321
…		…	
voxel 137	4	4	444

↑ Column 1: voxel values for sub-brick #2 from dataset **func_fred+tlrc** that fall within the ROI mask designated in dataset **ROI_fred+tlrc** (columns 2 and 3 are for sub-bricks #5 and #7 respectively)

- More than one dataset can be given at the end of the command line
- Main application of **3dmaskdump** is to dump out the data or functional values that match an ROI so they can be processed in some other program (e.g., Excel)
- If **-noijk** option is omitted, each output line starts with ijk-indexes (i.e., location) of the voxel
  - Program **3dUndump** can be used to create a dataset from a text file with ijk-indexes and dataset values

- **3dROIstats**
  - Program to compute average of voxels from a dataset, using multiple regions selected by a single ROI dataset
    - i.e., you can compute the mean for several ROIs separately and simultaneously
    - This differs from **3dmaskave** because the ROIs within a single mask are not collapsed and then averaged. Here the averages are done separately for each ROI within the mask dataset
  - Averaging is done over each region defined by a distinct value in the ROI dataset

★ Example:

```
3dROIstats -mask ROI_fred+tlrc  'func_fred+tlrc'
```

Output shown in the shell (use **>** command to save into to a text file):

File	Sub-brick	Mean_1	Mean_2	Mean_3
func_fred+tlrc	0	33.3	44.4	55.5
func_fred+tlrc	1	22.2	66.6	-21.2
func_fred+tlrc	2	2.3	9.7	777.777

➥ The **Mean_1** column is the average over the ROI whose mask value is "1". The average is calculated for voxels from our functional dataset **func_fred+tlrc**, that fall within the ROI. Averages are computed at each sub-brick (#0, #1, #2, etc)

⇨ Means have also been computed for ROIs whose mask value is "2" (Mean_2) and "3" (Mean_3)

★ Very useful if you creat ROI masks for a number of subjects, using the same number codes for the same anatomical regions (e.g., 1=hippocampus, 2=amygdala, 3=superior temporal gyrus, etc.)

★ You can load the output of **3dROIstats** into a spreadsheet for further analysis (e.g., statistics with other subjects' data)

Creating ROI datasets from Activation Maps

- The program **3dmerge** can find contiguous supra (above) threshold voxel clusters in an activation (functional) map and then convert each cluster into a ROI with a separate data value
  - ★ These ROIs can then be used as starting points for some analysis

- Example:

```
3dmerge -1clust_order 1 500
        -1tindex 1 -1thresh 0.4
        -prefix ROI_func
         func+tlrc
```

  - ★ **-1clust_order 1 500**: Here we've told the program to include voxels as part of a "cluster" if they are no more than 1mm apart. The 500 indicates the minimum volume required to form a cluster. In this example, a cluster is defined by 500 or more voxels grouped together, and each voxel is no more than 1 mm apart from each other
  - ★ **-1thresh 0.4**: Ignore voxels that don't survive your threshold (e.g., 0.4) -- the threshold could be any stat you have set, a t-test, an F-value, a correlation coefficient, a mean, etc..

The result:

- ★ Thresholds the dataset on subrick #1 (**`-1tindex 1`**) at value 0.4 (**`-1thresh 0.4`**)
- ★ Clusters together all the surviving nonzero voxels using a contiguity test of 1mm and keeping only clusters at least 500 mm^3 in volume (-1custorder 1 500)
- ★ All voxels in the largest cluster are assigned value 1 (say, 1200 voxels in one cluster), the second largest are assigned value 2 (say, 920 voxels in one cluster), etc., and the result is written to disk in dataset **`ROI_func+tlrc`**

- You can use this dataset as a mask, edit it with the drawing plugin, etc…