Details About Creating a Nipype Pipeline Notebook:

The Jupyter notebook code provided outlines a **hands-on tutorial** for creating a complete fMRI preprocessing workflow using **Nipype**. Below are the key points and steps covered in the notebook:

Overview

The notebook demonstrates how to:

1. **Set up a preprocessing pipeline** for fMRI data using various Nipype interfaces.
2. **Create and connect nodes** representing different preprocessing steps.
3. **Run the workflow** in parallel for efficient execution.
4. **Visualize the workflow** and inspect its outputs.

Preprocessing Workflow Structure

The preprocessing workflow consists of the following steps:

1. **Gunzip**: Unzips the functional image file.
2. **Drop Dummy Scans**: Removes initial scans to account for scanner instability.
3. **Slice Time Correction**: Corrects for differences in slice acquisition times.
4. **Motion Correction**: Corrects for subject motion during scanning.
5. **Artifact Detection**: Detects motion and intensity outliers.
6. **Segmentation**: Segments anatomical images into gray matter, white matter, and CSF.
7. **Coregistration**: Aligns functional images to anatomical images.
8. **Smoothing**: Applies Gaussian smoothing to the functional images.
9. **Apply Binary Mask**: Applies a mask to the smoothed images.
10. **Remove Linear Trends**: Removes linear trends from the time series data.

1. Node Creation and Workflow Setup

- The workflow is created using Nipype's Node and Workflow objects, which represent individual processing steps and the overall pipeline, respectively.

python

from nipype **import** Node, Workflow

preproc = Workflow(name='work_preproc', base_dir='/output/')

Example Node Creation (Gunzip):

python

from nipype.algorithms.misc **import** Gunzip

gunzip_func = Node(Gunzip(in_file=func_file), name='gunzip_func')

2. Connecting Nodes

- Each node is connected to form a pipeline that defines the order of execution.

python

preproc.connect([(gunzip_func, extract, [('out_file', 'in_file')])])

3. Preprocessing Steps

Slice Time Correction

- Uses SPM's SliceTiming interface to correct for slice acquisition timing differences.

python

from nipype.interfaces.spm **import** SliceTiming

slicetime = Node(SliceTiming(num_slices=30, ref_slice=15, slice_order=slice_order, time_repetition=2.5), name='slicetime')

preproc.connect([(extract, slicetime, [('roi_file', 'in_files')])])

Motion Correction

- Uses FSL's MCFLIRT interface to correct for motion artifacts.

python

from nipype.interfaces.fsl **import** MCFLIRT

mcflirt = Node(MCFLIRT(mean_vol=True, save_plots=True), name="mcflirt")

preproc.connect([(slicetime, mcflirt, [('timecorrected_files', 'in_file')])])

Artifact Detection

- Detects motion and intensity outliers using Nipype's ArtifactDetect.

python

```
from nipype.algorithms.rapidart import ArtifactDetect

art = Node(ArtifactDetect(norm_threshold=0.6, zintensity_threshold=2.2), name="art")

preproc.connect([(mcflirt, art, [('out_file', 'realigned_files'), ('par_file', 'realignment_parameters')])])
```

Segmentation

- Segments an anatomical image into gray matter (GM), white matter (WM), and CSF using SPM's NewSegment.

python

```
from nipype.interfaces.spm import NewSegment

segment = Node(NewSegment(tissues=tissues), name='segment')

preproc.connect([(gunzip_anat, segment, [('out_file', 'channel_files')])])
```

Coregistration

- Aligns functional images with anatomical images using FSL's FLIRT.

python

```
from nipype.interfaces.fsl import FLIRT

coreg = Node(FLIRT(dof=6, cost='bbr'), name="coreg")

preproc.connect([(mcflirt, coreg, [('mean_img', 'in_file')])])
```

Smoothing

- Smooths functional images using FSL's SUSAN workflow.

python

```
from niflow.nipype1.workflows.fmri.fsl.preprocess import create_susan_smooth

susan = create_susan_smooth(name='susan')

preproc.connect([(applywarp, susan, [('out_file', 'inputnode.in_files')])])
```

4. Data Input with SelectFiles and Iterables

- The SelectFiles node is used to automatically select input files from a BIDS-formatted dataset.

python

```
from nipype import SelectFiles

sf = Node(SelectFiles(templates), name='selectfiles')

sf.iterables = [('subject_id', subject_list)]

preproc.connect([(sf, gunzip_anat, [('anat', 'in_file')]), (sf, gunzip_func, [('func', 'in_file')])])
```

5. Visualizing the Workflow

- The workflow graph is generated and visualized within the notebook to understand how data flows between nodes.

python

```
preproc.write_graph(graph2use='colored', format='png', simple_form=True)

Image(filename='/output/work_preproc/graph.png')
```

6. Running the Workflow

- The workflow is run in parallel using multiple processors (specified by n_procs) to speed up execution.

python

```
preproc.run('MultiProc', plugin_args={'n_procs': 4})
```

7. Output Inspection

- After running the workflow, the outputs are stored in directories corresponding to each step (e.g., gunzip, mcflirt, susan). The results can be inspected using standard file system commands.

bash

```
!tree /output/work_preproc -I '*js|*json|*pklz|_report|*.dot|*html'
```

Conclusion

This notebook provides a comprehensive guide on how to:

1. Set up an fMRI preprocessing pipeline using Nipype.
2. Create and connect nodes for each preprocessing step (e.g., slice timing correction, motion correction).

3. Run workflows in parallel for efficient processing.
4. Visualize workflows and inspect outputs.

Nipype's modular design allows researchers to build reproducible neuroimaging pipelines that integrate various tools (e.g., FSL, SPM) while leveraging parallel computing for faster execution times.