In [28]:
```
#import sys
#sys.path.insert(1, '/users/juusokor/.local/lib/python3.9/site-packages')
#sys.path.remove('/scratch/project_2003855/jupyter_scripts/pypyls-master')
#sys.path
#sys.path.append('/users/juusokor/.local/lib/python3.9/site-packages')
%reset -f
```

Table of Contents

Introduction

Psychopathy is an antisocial personality disorder that is highly correlated with crime and violence. Psychopathy Check List Revised (PCL-R) provides a clinical assessment of the degree of psychopathy that an individual possesses. Specific scoring criteria rate twenty separate items on a three-point scale (0, 1, 2) to determine the extent to which they apply to a given individual. In this analysis we do Partial-Least-Squares (PLS) analysis on sMRI images of PCL-R assessed forensic patients, to see if there exists reliable biomarkers that correlate with the subscores of the PCL-R. Sample size is 51. As preprocessing, grey matter (GM) images are segmented from the sMRI images with SPM software. Then, regional volumes are calculated from the GM images with masks provided also by the SPM software. These regional volumes are then normalized with total GM volume, so that we analyze volumes relative to individuals total GM volume instead of analyzing absolute volumes.

PLS aims to find latent components (linear combinations) from both, volume features X and behavior measures Y, that maximally correlate. This is achieved by doing Singular-Value-Decomposition (SVD) on the correlation matrix R which is calculated from normalized X (X0) and normalized Y (Y0) as R = transpose(Y0) * X0.

Load and normalize the data

In [29]:
```
import pandas as pd
import numpy as np
import nibabel as nib

metadata = pd.read_csv('/scratch/project_2003855/data/NIUVA/METADATA/PCL_R.csv')
#metadata.head()
img_size = 121*145*121
n_samp = 51
voxels = np.zeros([n_samp, img_size])

for filepath, i in zip(metadata["sGM_filepath_csc"], range(n_samp)):
    # Load 3D image
    fp = filepath.replace("smwp1", "mwp1") # unsmoothed
    img_nifti = nib.load(fp)
    # Get voxel values
    img = img_nifti.get_fdata()
    # Vectorize to 1D and store in X
    voxels[i, :] = img.flatten()

# NORMALIZATION!!! this is very critical for the results
from sklearn.preprocessing import normalize

voxels = normalize(voxels, norm='l1', axis=1) # default is row-wise L2-normalization
```

In [30]:
```
from scipy import ndimage
import nibabel as nib
import numpy as np
import matplotlib.pyplot as plt
from ipywidgets import interact, interactive, fixed, interact_manual
from matplotlib.colors import LogNorm
from scipy.ndimage import zoom
import xml.etree.ElementTree as ET
# Load the SPM atlas to get masks for different brain regions
atlas = nib.load('/scratch/project_2003855/jupyter_scripts/atlasData/labels_Neuromorphometrics.nii')
# Get voxel values
atlas = atlas.get_fdata()
# Vectorize 3D image
temp = atlas.flatten()
region_ii = np.unique(atlas)[1:] # 0 is background
n_region = len(region_ii)
# get region names from separate xml file
tree = ET.parse('/scratch/project_2003855/jupyter_scripts/atlasData/ROIlabels.xml')
root = tree.getroot()
region_names = [root[int(i)][1].text for i in range(n_region)]
Regional_Volumes = pd.DataFrame(data=np.zeros([51, n_region]), columns=region_names)

for region_i, i in zip(region_ii, range(n_region)):
    Regional_Volumes.iloc[:, i] = np.sum(voxels[:,temp==region_i], axis=1)

plot_atlas = 0
if plot_atlas:
    # examples of how areas are labeled in atlas (first is left, and then right)
    # amygdala:
    # atlas==31 | atlas==32
    # orbitofrontal cortex:
    # atlas==104 | atlas==105 ...
        #| atlas==146 | atlas==147
    # anterior cingulate gyrus:
    # atlas==100 | atlas==101
    # hippocampus:
    # atlas==47 | atlas==48
    region = 32
    wholemask = atlas.copy()
    wholemask[wholemask != region] = 0

    def explore_3d_image(layer):
        plt.figure(figsize=(10, 15))
        im = ndimage.rotate(atlas[:, :,layer], -90)
        mask = ndimage.rotate(wholemask[:,:,layer], -90)
        plt.imshow(im, cmap='gray', vmin=0, vmax=255)
        plt.imshow(mask, cmap='coolwarm', vmin=-1, vmax=1, alpha=0.5)
        plt.axis('off')
        plt.colorbar(fraction=0.046, pad=0.04)
        return layer

    interact(explore_3d_image, layer=(0, 120))
```

Fill the missing subscores according to total score (even distribution)

In [31]:
```
Y = metadata.iloc[:, 38:59]
#print(np.where(Y.iloc[row,:].isna())[0][1])
for row in np.where(Y.isna().any(axis=1))[0]:
    #print(row)
    # Calculate missing part of the total score
    m = Y.iloc[row, 0] - np.nansum(Y.iloc[row, 1:])
    missing = np.where(Y.iloc[row,:].isna())[0]
    # Divide the missing part equally to Nans
    #print(len(missing))
    fillval = np.ceil(m / len(missing))
    Y.iloc[row, missing] = fillval

#from pandas import DataFrame

#DataFrame(np.array([Y["PCL_R_tot"], np.sum(Y.iloc[:, 1:], axis=1), Y["PCL_R_tot"] - np.sum(Y.iloc[:, 1:], axis=1)]).T)
```

In [32]:
```
X = Regional_Volumes
# Remove total score
Y = Y.iloc[:,1:]
subscore_names = Y.columns
print("Shape of X: ", np.shape(X))
print("Shape of Y: ", np.shape(Y))
print("Behaviors: ", subscore_names)
```

```
Shape of X:  (51, 136)
Shape of Y:  (51, 20)
Behaviors:  Index(['1. Glibness/Superficial Charm', '2. Grandiose Sense of Self Worth',
       '3. Need for Stimulation/Proneness to Boredom', '4. Pathological Lying',
       '5. Conning/Manipulative', '6. Lack of Remorse of Guilt',
       '7. Shallow Affect', '8. Callous/Lack of Empathy',
       '9. Parasitic Lifestyle', '10. Poor Behavioral Controls',
       '11. Promiscuous Sexual Behavior', '12. Early Behavioral Problems',
       '13. Lack of Realistic, Long-term Goals', '14. Impulsivity',
       '15. Irresponsibility',
       '16. Failure to Accept Responsibility for Own Actions',
       '17. Many Short-term Marital Relationships', '18. Juvenile Delinquency',
       '19. Revocation of Conditional Release', '20. Criminal Versatility'],
      dtype='object')
```

Or load the data as numpy arrays:

In [82]:
```
import numpy as np

from_saved = True

if from_saved:
    #X = np.genfromtxt('/scratch/project_2003855/jupyter_scripts/Brain.csv',delimiter=',')
    X = np.genfromtxt('/scratch/project_2003855/jupyter_scripts/numpyData/RegionalVolumesTotalGMnormed.csv', delimiter=',', skip_header=True)
    Y1 = np.genfromtxt('/scratch/project_2003855/jupyter_scripts/numpyData/F1.csv',delimiter=',', skip_header=True) # Factor 1
    F1 = np.sum(Y1, axis=1, keepdims=True) # F1 total
    Y2 = np.genfromtxt('/scratch/project_2003855/jupyter_scripts/numpyData/F2.csv',delimiter=',', skip_header=True) # Factor 2
    F2 = np.sum(Y2, axis=1, keepdims=True) # F2 total
    covars = pd.read_csv('/scratch/project_2003855/jupyter_scripts/numpyData/CofoundingVariables.csv') #possibly cofounding variables
    covars_bin = pd.get_dummies(covars)
    Y = np.concatenate((Y1,Y2), axis = 1)
    #Y = np.concatenate((F1,Y1,F2,Y2, covars_bin), axis = 1)
    #Y = np.concatenate((F1,F2), axis = 1)
    #Y = Y1
    #Y = Y2
    print(X.shape)
    print(Y.shape)

    # Load the subscore label names separately through Pandas:
    import pandas as pd
    from IPython.display import display, HTML
    temp = pd.read_csv('/scratch/project_2003855/jupyter_scripts/numpyData/F1.csv')
    temp2 = pd.read_csv('/scratch/project_2003855/jupyter_scripts/numpyData/F2.csv')
    subscore_names = np.concatenate((list(temp.columns),list(temp2.columns)), axis = 0)
    #subscore_names = np.concatenate((["F1"], list(temp.columns),["F2"], list(temp2.columns), list(covars_bin.columns)), axis = 0)
    #subscore_names = np.concatenate((["Factor 1 total"], list(temp.columns),["Factor 2 total"], list(temp2.columns)), axis = 0)
    #subscore_names = np.concatenate((["Factor 1 total"], ["Factor 2 total"]), axis = 0)
    #subscore_names
    lbls = pd.DataFrame({'Factor 1 labels': temp.columns, 'Factor 2 labels': temp2.columns})
    display(lbls)
else:
    temp = pd.read_csv('/scratch/project_2003855/jupyter_scripts/numpyData/F1.csv')
    temp2 = pd.read_csv('/scratch/project_2003855/jupyter_scripts/numpyData/F2.csv')
    F1 = np.sum(Y[temp.columns], axis=1)
    F2 = np.sum(Y[temp2.columns], axis=1)
    Y = pd.concat([F1, F2], axis=1)

    X = X.to_numpy()
    Y = Y.to_numpy()
```

```
(51, 136)
(51, 20)
```

	Factor 1 labels	Factor 2 labels
0	1. Glibness/Superficial Charm	3. Need for Stimulation/Proneness to Boredom
1	2. Grandiose Sense of Self Worth	9. Parasitic Lifestyle
2	4. Pathological Lying	10. Poor Behavioral Controls
3	5. Conning/Manipulative	11. Promiscuous Sexual Behavior
4	6. Lack of Remorse of Guilt	13. Lack of Realistic, Long-term Goals
5	7. Shallow Affect	14. Impulsivity
6	8. Callous/Lack of Empathy	15. Irresponsibility
7	16. Failure to Accept Responsibility for Own A...	18. Juvenile Delinquency
8	17. Many Short-term Marital Relationships	12. Early Behavioral Problems
9	20. Criminal Versatility	19. Revocation of Conditional Release

Normalize the input features X and output behaviors Y with standard scaling. Alternatively we could just not do this feature wise normalization, but then the differences in large regions would dominate.

In [84]:
```
from sklearn import preprocessing

scalerX = preprocessing.StandardScaler().fit(X)
X0 = scalerX.transform(X)

scalerY = preprocessing.StandardScaler().fit(Y)
Y0 = scalerY.transform(Y)
```

Or then we do the normalization scanner-wise in order to correct the scanner effect (not working):

In [85]:
```
"""
from sklearn import preprocessing
#possibly cofounding variables
if from_saved:
    covars = pd.read_csv('/scratch/project_2003855/jupyter_scripts/numpyData/CofoundingVariables.csv') #possibly cofounding variables
    covars.columns = ["SEX", 'ManufacturerModelName', "AGE"]
else:
    covars = metadata[["AGE", 'ManufacturerModelName']]
    # not legit, we don't know the ages of Niuva images
    covars.AGE = covars.AGE.fillna(30)
    # the niuva images are scanned with siemens scanner
    covars.ManufacturerModelName = covars.ManufacturerModelName.fillna("Siemens")

grouping = 0

if grouping:
    scanners = covars["ManufacturerModelName"]

    Xdf = pd.DataFrame(data=X)
    Xdf["group"] = scanners
    Xdf = Xdf.groupby('group').transform(lambda x: (x - x.mean()) / x.std())

    X0 = Xdf.to_numpy()

    Ydf = pd.DataFrame(data=Y)
    Ydf["group"] = scanners
    Ydf = Ydf.groupby('group').transform(lambda x: (x - x.mean()))


    Y0 = Ydf.to_numpy()
"""
```

Out[85]:
```
'\nfrom sklearn import preprocessing\n#possibly cofounding variables\nif from_saved:\n    covars = pd.read_csv(\'/scratch/project_2003855/jupyter_scripts/numpyDa
ta/CofoundingVariables.csv\') #possibly cofounding variables\n    covars.columns = ["SEX", \'ManufacturerModelName\', "AGE"]\nelse:\n    covars = metadata[["AG
E", \'ManufacturerModelName\']]\n    # not legit, we don\'t know the ages of Niuva images\n    covars.AGE = covars.AGE.fillna(30)\n    # the niuva images are sca
nned with siemens scanner\n    covars.ManufacturerModelName = covars.ManufacturerModelName.fillna("Siemens")\n\ngrouping = 0\n\nif grouping:\n    scanners =
covars["ManufacturerModelName"]\n\n    Xdf = pd.DataFrame(data=X)\n    Xdf["group"] = scanners\n    Xdf = Xdf.groupby(\'group\').transform(lambda x: (x - x.mean
()) / x.std())\n\n    X0 = Xdf.to_numpy()\n\n    Ydf = pd.DataFrame(data=Y)\n    Ydf["group"] = scanners\n    Ydf = Ydf.groupby(\'group\').transform(lambda x: (x
- x.mean()))\n\n    Y0 = Ydf.to_numpy()\n'
```

Run the PLS-analysis

Run the PLS-analysis:

In [86]:
```
# run the analysis and look at the results structure
from pyls import behavioral_pls
# grouped option takes the cofounding scanner effect into consideration
grouped = 0
if grouped:
    if from_saved:
        covars = pd.read_csv('/scratch/project_2003855/jupyter_scripts/numpyData/CofoundingVariables.csv') #possibly cofounding variables
        covars.columns = ["SEX", 'ManufacturerModelName', "AGE"]
    else:
        covars = metadata[["AGE", 'ManufacturerModelName']]
        # not legit, we don't know the ages of Niuva images
        covars.AGE = covars.AGE.fillna(30)
        # the niuva images are scanned with siemens scanner
        covars.ManufacturerModelName = covars.ManufacturerModelName.fillna("Siemens")

    scanners = covars["ManufacturerModelName"]

    Xdf = pd.DataFrame(data=X)
    Xdf["group"] = scanners
    Xdf = Xdf.sort_values("group")
    Xdf = Xdf.drop("group", axis=1)
    X0 = Xdf.to_numpy()

    Ydf = pd.DataFrame(data=Y)
    Ydf["group"] = scanners
    Ydf = Ydf.sort_values("group")
    groups = Ydf.groupby("group").size().values
    Ydf = Ydf.drop("group", axis=1)
    Y0 = Ydf.to_numpy()


    print(groups)
    bpls = behavioral_pls(X0, Y0, groups=groups, n_proc='max', verbose=False)
else:
    bpls = behavioral_pls(X0, Y0, n_proc='max', test_split=0, test_size=0, verbose=False, seed=1234)
bpls
```

Out[86]:
```
PLSResults(x_weights, y_weights, x_scores, y_scores, y_loadings, singvals, varexp, permres, bootres, inputs)
```

Evaluate PLS-analysis results

Let see how much of the covariance (variance of R) is captured by our latent components:

In [87]:
```
bpls.varexp
```

Out[87]:
```
array([0.52334593, 0.09529774, 0.07468434, 0.05659722, 0.04730812,
       0.03930069, 0.02748094, 0.02501693, 0.02030937, 0.01692643,
       0.01517063, 0.01384114, 0.00949829, 0.00866271, 0.00762415,
       0.00553924, 0.00539246, 0.00369562, 0.0026619 , 0.00164615])
```

Permutation testing over singular values: Which components are significant?

During the analysis, permutation testing is done in order to evaluate whether the signal is different from noise. This is done through shuffling the labels, doing the PLS analysis again and seeing how much variance is captured with the newly created latent components. This is done 5000 times to create a noise distribution, from which the p-value is then calculated for the original values of the captured variance.

In [88]:
```
permres = bpls.get("permres")
permres.get("pvals")
```

Out[88]:
```
array([1.99960008e-04, 6.99860028e-03, 4.13317337e-01, 3.32133573e-01,
       2.72745451e-01, 7.71445711e-01, 7.90441912e-01, 5.71485703e-01,
       9.40811838e-01, 9.88002400e-01, 9.76404719e-01, 9.70605879e-01,
       9.51609678e-01, 9.71805639e-01, 9.96200760e-01, 9.96200760e-01,
       1.00000000e+00, 1.00000000e+00, 9.99800040e-01, 1.00000000e+00])
```

After correcting the p-values for multiple comparison (feature size 136), we are left with 1 significant latent component (1st latent component).

Posthoc t-tests on PLS scores: Are components differently expressed by groups?

We do posthoc t-tests on x-scores between psychopaths (total score > 25) and non-psychopaths (total score <= 25). The signifigance of the results is questionable since the components were already discovered using subscores.

In [89]:
```
from scipy import stats

PCL_R_tot_score = np.sum(Y, axis=1)
is_psycho = (PCL_R_tot_score > 25)
stats.ttest_ind(bpls.x_scores[is_psycho==1,0], bpls.x_scores[is_psycho==0,0], equal_var=False)
```

Out[89]:
```
Ttest_indResult(statistic=-3.8161014317433555, pvalue=0.0004286981228366556)
```

Significant Latent Component analysis

Lets check the pearson correlation between the x-scores (points in the volume coordinate space reflected onto the imaging latent component) and the y-scores (points in the behavior measure space reflected onto the respective behavioral latent component).

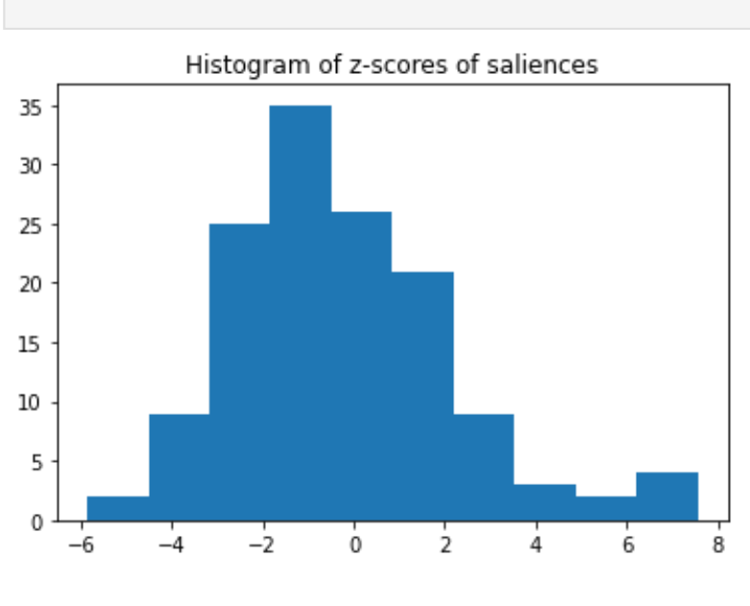
In [90]:
```
from scipy.stats import pearsonr
lc = 0
pearsonr(bpls.x_scores[:,lc], bpls.y_scores[:,lc])[0]
```

Out[90]:
```
0.7478759356555557
```

Bootstrap estimation of saliences' standard errors: Which PLS loadings are stable?

adjustment (usually distorted by choice of standard)Imaging saliences can be concidered as the coefficients for the original imaging variables to construct the latent component. To estimate the standard errors of the saliences, we create bootstrap samples which are obtained by sampling with replacement from X. A salience standard error is then estimated as the standard error of the saliences from a large number of these bootstrap samples (5000 in our case). The ratios (salience / standard error of the salience) are akin to a Z-score, therefore when they are larger than 2 the corresponding saliences are considered significantly stable. Stable saliences determine which imaging variables show reliable responses to the behavioral measures (across the population, not dependant on outliers).

In [91]:
```
import matplotlib.pyplot as plt
plt.hist(bpls.bootres.x_weights_normed[:,lc])
plt.title("Histogram of z-scores of saliences")
plt.show()
```



Driving imaging features

Which imaging variables drive the latent component: pearson correlation between the original variable scores and the latent component scores.

In [92]:
```
from scipy.stats import pearsonr

imageLoadings = np.zeros([X.shape[1],])

for i in range(X.shape[1]):
    imageLoadings[i] = pearsonr(X[:,i], bpls.x_scores[:,lc])[0]
```

Important imaging features are:

```
- stable (according to bootstrap results)

- correlate with the latent component (pearson correlation > 0.7)
```

In [93]:
```
np.where((abs(imageLoadings) > 0.5) & (abs(bpls.bootres.x_weights_normed[:,lc]) > 2))
```

Out[93]:
```
(array([ 14,  17,  28,  31,  50,  55,  57,  73,  74,  85,  86,  87,  94,
        107, 113, 117, 131, 132]),)
```

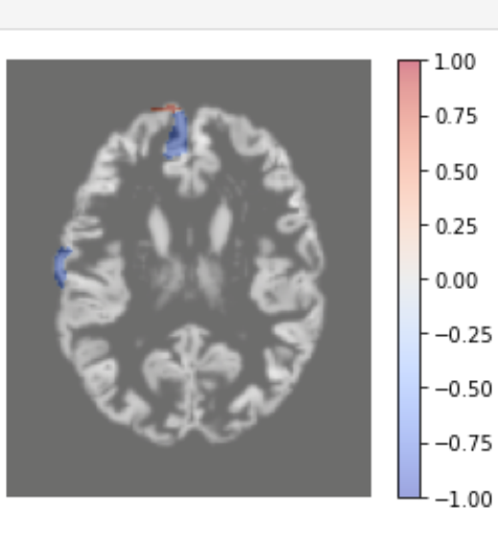
Lets translate these indexes as brain region labels found in the original xml file of the SPM software:

In [94]:
```
import xml.etree.ElementTree as ET
tree = ET.parse('/scratch/project_2003855/jupyter_scripts/atlasData/ROIlabels.xml')
root = tree.getroot()

idxs = np.where((abs(imageLoadings) > 0.7) & (abs(bpls.bootres.x_weights_normed[:,lc]) > 2))
for i in idxs[0]:
    print(root[i][1].text)
```

```
Left FRP frontal pole
Left MSFG superior frontal gyrus medial segment
Right OrIFG orbital part of the inferior frontal gyrus
Left PoG postcentral gyrus
```

Lets further plot the correlations of stable regions as mask on top of single subject MRI image (template image from SPM software):

In [110…
```
import nibabel as nib
import numpy as np
import matplotlib.pyplot as plt
from ipywidgets import interact, interactive, fixed, interact_manual
from matplotlib.colors import LogNorm
from scipy.ndimage import zoom

def read_nifti_file(filepath):
    """Read and load volume"""
    # Read file
    scan = nib.load(filepath)
    # Get raw data
    scan = scan.get_fdata()
    return np.fliplr(scan)

image_data = read_nifti_file('/scratch/project_2003855/jupyter_scripts/atlasData/labels_Neuromorphometrics.nii')
loadings = np.zeros_like(image_data)

zs = bpls.bootres.x_weights_normed[:,lc]

zscores = np.zeros_like(image_data)

region_idx = np.unique(image_data)[1:] # first is background

for i in range(np.size(region_idx)):
    loadings[image_data==region_idx[i]] = imageLoadings[i]
    zscores[image_data==region_idx[i]] = zs[i]


data = loadings
data[np.abs(zscores) < 2] = 0
data[np.abs(data) < 0.7] = 0

single_subj = read_nifti_file(fp)#read_nifti_file('/scratch/project_2003855/jupyter_scripts/atlasData/single_subj_T1.nii') #
#print(single_subj.shape)
#print(image_data.shape)
h,w,d = single_subj.shape

single_subj = zoom(single_subj, [121/h, 145/w, 121/d])
#print(single_subj.shape)

from scipy import ndimage

def explore_3d_image(layer):
    plt.figure(figsize=(10, 15))
    channel = 3
    im = ndimage.rotate(single_subj[:, :,layer], -90)
    mask = ndimage.rotate(data[:,:,layer], -90)
    plt.imshow(im, cmap='gray')
    plt.imshow(-mask, cmap='coolwarm', vmin=-1, vmax=1, alpha=0.5)
    plt.axis('off')
    plt.colorbar(fraction=0.046, pad=0.04)
    return layer

#interact(explore_3d_image, layer=(0, 120))
plot_single = True
if plot_single:
    im = ndimage.rotate(single_subj[:, :,60], -90)
    mask = ndimage.rotate(data[:,:,60], -90)
    plt.imshow(im, cmap='gray')
    plt.imshow(-mask, cmap='coolwarm', vmin=-1, vmax=1, alpha=0.5)
    plt.axis('off')
    plt.colorbar(fraction=0.046, pad=0.04)
```



In the plot, warm areas (red) are positively correlated with behavior scores, which is to say that the GM volume tends to increase with the behavior scores in these areas. Cold areas (blue) are negatively correlated, which is to say that GM volume tends to decrease in these areas with increase in behavior scores.

(It is little bit more complicated as latent variable could be directed in negative and positive directions for different subscores, but mostly they are the same [as can be seen in the next section])

Driving behavioral measures

We can determine driving behavioral measures by correlating the original measures with the x-scores. Important behavioral measures are:

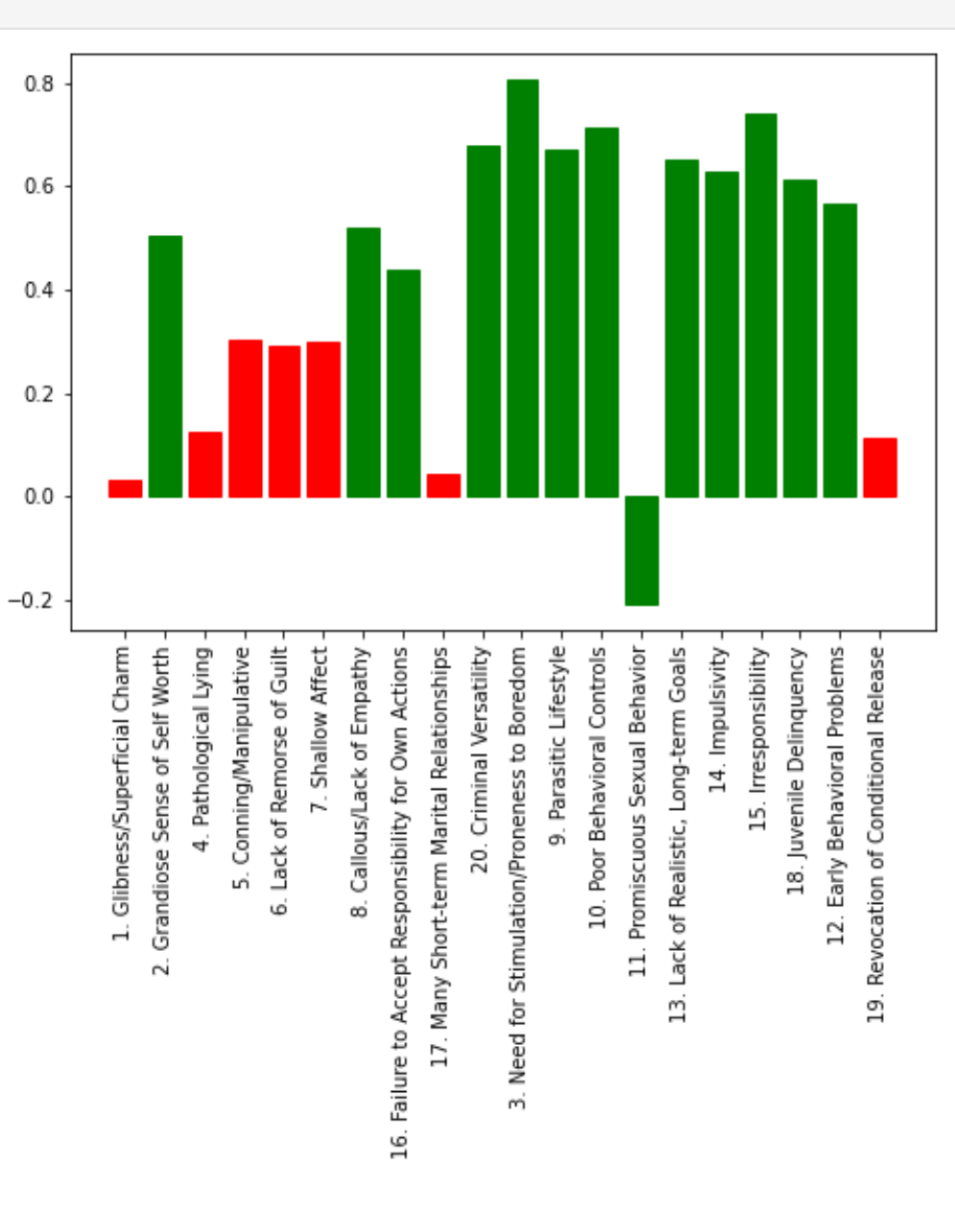
```
- stable (bootstrap confidence interval of correlation is narrow)
- good correlation with x-scores
```

Correlations came out from the analysis as opposite sign, but for clarifity they are switched along with the correlations of image regions!

In [96]:
```
np.shape(behaviors)
```

Out[96]:
```
(20,)
```

In [105…
```
# Which behavioral measures drive components: correlation between original variables and components
from scipy.stats import pearsonr

behaviorLoadings = np.zeros([Y.shape[1],])
for i in range(Y.shape[1]):
    behaviorLoadings[i] = pearsonr(Y[:,i], bpls.y_scores[:,lc])[0]

means = -behaviorLoadings #-bpls.bootres.y_loadings[:,0] #behaviorLoadings
#print(means)
conf  = -bpls.bootres.y_loadings_ci[:,lc]
#print(conf)
# SE: (upper limit - lower limit) / 3.92
se = (conf[:,1] - conf[:,0]) / 3.92

#plt.hist(means/se)

stable = (abs(-bpls.bootres.y_loadings[:,lc]/se) > 2)*1
#print(stable)
# calculate the error
yerr = np.c_[means-conf[:,0],conf[:,1]-means ].T

fig = plt.figure()
ax = fig.add_axes([0,0,1,1])

behaviors = subscore_names# ["F1", "F2"]#subscore_names
b = ax.bar(behaviors, means)
#b[stable.to_byteslist()].set_color('green')

#[b[i].set_color('r') for i in stable if i==0]


for i in range(len(stable)):
    b[i].set_color('r')
    if stable[i] == 1:
        b[i].set_color('g')

y_pos = range(len(behaviors))

plt.xticks(y_pos, behaviors, rotation=90)
plt.show()
```



Cofounding Variable Analysis

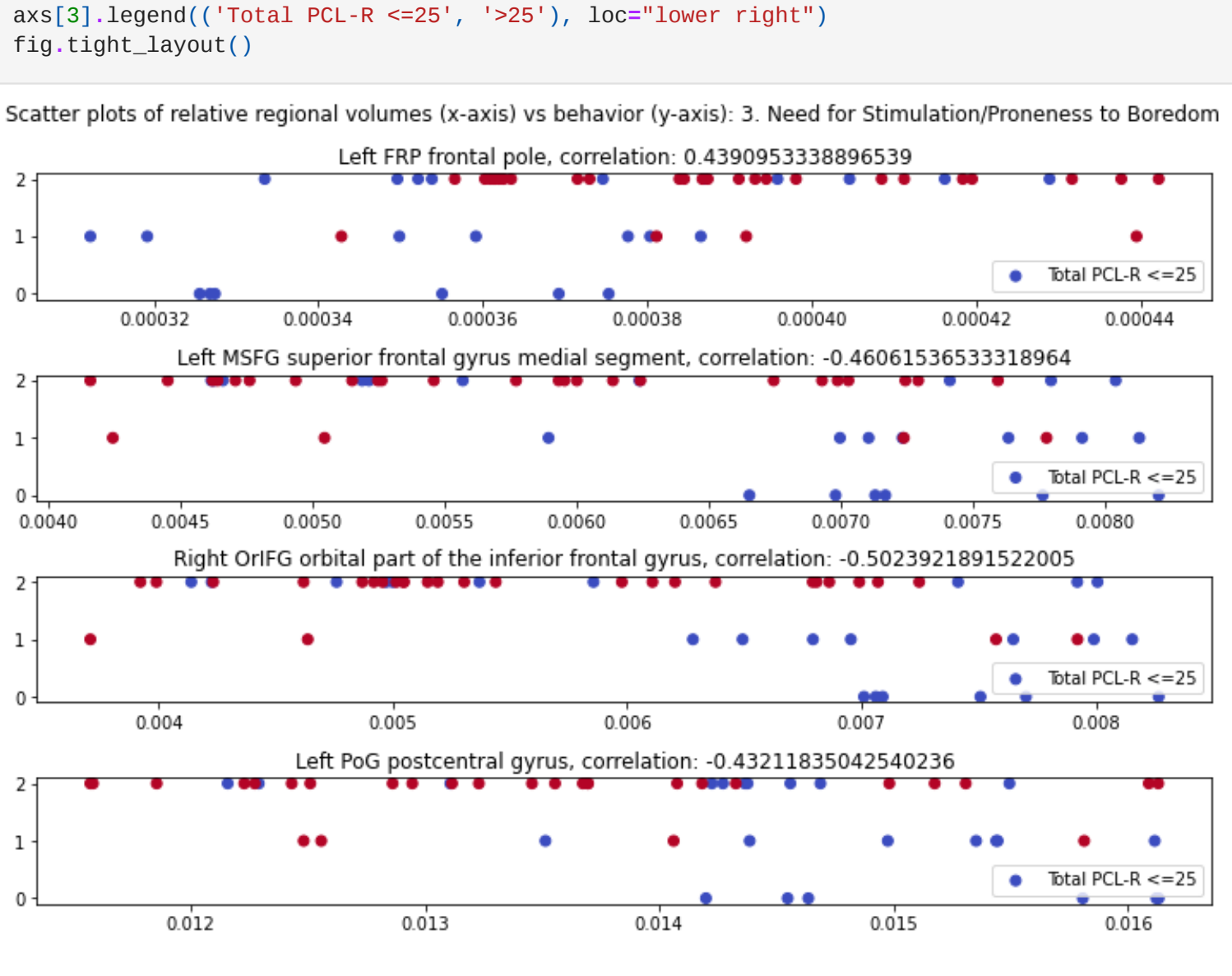
In [99]:
```
if from_saved:
    covars = pd.read_csv('/scratch/project_2003855/jupyter_scripts/numpyData/CofoundingVariables.csv') #possibly cofounding variables
    covars.columns = ["SEX", 'ManufacturerModelName', "AGE"]
else:
    covars = metadata[["AGE", 'ManufacturerModelName']]
    # not legit, we don't know the ages of Niuva images
    covars.AGE = covars.AGE.fillna(30)
    # the niuva images are scanned with siemens scanner
    covars.ManufacturerModelName = covars.ManufacturerModelName.fillna("Siemens")

#print(covars.columns)
xscores = bpls.x_scores[:,lc]
yscores = bpls.y_scores[:,lc]

agecorx = pearsonr(covars["AGE"], xscores)[0]
agecory = pearsonr(covars["AGE"], yscores)[0]
#print("Correlation of age with:\n\tx-scores: " + str(agecorx))
#print("\ty-scores: " + str(agecory) + "\n")

scanners = covars['ManufacturerModelName']
scannerModels = np.unique(scanners)

for model in scannerModels:
    ismodel = (scanners == model)*1
    ismodel = ismodel.to_numpy()
    print("Correlation of being a model " + str(model) + " with:")
    print("\tx-scores: " + str(pearsonr(ismodel, xscores)[0]))
    print("\ty-scores: " + str(pearsonr(ismodel, yscores)[0]))
    print("")
```

```
Correlation of being a model Avanto with:
        x-scores: 0.27329103880582545
        y-scores: 0.1991750830220575

Correlation of being a model SIGNA Voyager with:
        x-scores: 0.362492797565558
        y-scores: 0.3297997708950452

Correlation of being a model Siemens with:
        x-scores: -0.8554689960481457
        y-scores: -0.6474892552440064

Correlation of being a model Signa HDxt with:
        x-scores: 0.5085855570508178
        y-scores: 0.34640336470198235
```

In [100…
```
# if you want to save the results
save = 0
extension = "_F1F2.csv"
if save==1:
    imres = pd.DataFrame({"imgLoadings" : imageLoadings, "zscores" : bpls.bootres.x_weights_normed[:,1]})
    imres.to_csv("/scratch/project_2003855/jupyter_scripts/ImgRes" + extension)
    bhvres = pd.DataFrame({"bhvLoadings" : behaviorLoadings}) # z-scores should be also calculated
    bhvres.to_csv("/scratch/project_2003855/jupyter_scripts/BhvRes" + extension)
```

Plot the relative volumes of brain regions found important with respect to behavior scores:

In [107…
```
from matplotlib.pyplot import scatter

roi_ii = np.where((abs(imageLoadings) > 0.7) & (abs(bpls.bootres.x_weights_normed[:,lc]) > 2))[0]
# choose the behavior
beh_idx = 10
behavior = Y[:,beh_idx]

PCL_R_tot_score = np.sum(Y, axis=1)
is_psycho = (PCL_R_tot_score > 25)

%precision %.2f

fig, axs = plt.subplots(4, figsize=(10,7))
fig.suptitle('Scatter plots of relative regional volumes (x-axis) vs behavior (y-axis): ' + behaviors[beh_idx])
axs[0].scatter(X[:,roi_ii[0]], behavior, c = is_psycho, cmap="coolwarm")
axs[0].title.set_text(root[roi_ii[0]][1].text + ", correlation: " + str(pearsonr(X[:,roi_ii[0]], behavior)[0]))
axs[0].legend(('Total PCL-R <=25', '>25'), loc="lower right")
axs[1].scatter(X[:,roi_ii[1]], behavior, c = is_psycho, cmap="coolwarm")
axs[1].title.set_text(root[roi_ii[1]][1].text + ", correlation: " + str(pearsonr(X[:,roi_ii[1]], behavior)[0]))
axs[1].legend(('Total PCL-R <=25', '>25'), loc="lower right")
axs[2].scatter(X[:,roi_ii[2]], behavior, c = is_psycho, cmap="coolwarm")
axs[2].legend(('Total PCL-R <=25', '>25'), loc="lower right")
axs[2].title.set_text(root[roi_ii[2]][1].text + ", correlation: " + str(pearsonr(X[:,roi_ii[2]], behavior)[0]))
axs[3].scatter(X[:,roi_ii[3]], behavior, c = is_psycho, cmap="coolwarm")
axs[3].title.set_text(root[roi_ii[3]][1].text + ", correlation: " + str(pearsonr(X[:,roi_ii[3]], behavior)[0]))
axs[3].legend(('Total PCL-R <=25', '>25'), loc="lower right")
fig.tight_layout()
```



Quick conclusions:

There seems to be a moderate negative correlation between the relative volumes of the following brain regions and the following behavior scoring:

In [108…
```
idxs = np.where(((imageLoadings) > 0.7) & ((bpls.bootres.x_weights_normed[:,lc]) > 2))
print("Brain regions: ")
for i in idxs[0]:
    print("\t" + root[i][1].text)

print("Behaviors: ")
[print("\t" + lbl) for lbl in subscore_names[behaviorLoadings < -0.7]]
```

```
Brain regions:
        Left MSFG superior frontal gyrus medial segment
        Right OrIFG orbital part of the inferior frontal gyrus
        Left PoG postcentral gyrus
Behaviors:
        3. Need for Stimulation/Proneness to Boredom
        10. Poor Behavioral Controls
        15. Irresponsibility
```

Out[108…
```
[None, None, None]
```

There seems to be a moderate positive correlation between the relative volumes of the following brain regions and the following behavior scoring:

In [109…
```
idxs = np.where(((imageLoadings) < -0.7) & ((bpls.bootres.x_weights_normed[:,lc]) < -2))
print("Brain regions: ")
for i in idxs[0]:
    print("\t" + root[i][1].text)

print("Behaviors: ")
[print("\t" + lbl) for lbl in subscore_names[behaviorLoadings < -0.7]]
```

```
Brain regions:
        Left FRP frontal pole
Behaviors:
        3. Need for Stimulation/Proneness to Boredom
        10. Poor Behavioral Controls
        15. Irresponsibility
```

Out[109…
```
[None, None, None]
```

The findings however are cofounded by the used scanner (especially by the use of Siemens scanner). This means we can't conclude if the differences are due to the antisocial behavior or due to different scanners being used.