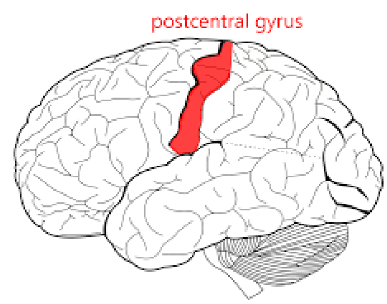
CSAI - ASSIGNMENT 1

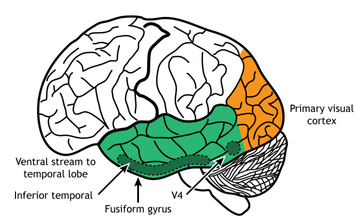
I am Prisha, 2021101075. I've been assigned the ROI as Post Central and X as bottle with subject 2.

POST CENTRAL



The post-central cortex, part of the somatosensory system, is primarily responsible for processing tactile and proprioceptive information. It is not directly specialized for visual processing, which limits its ability to differentiate between complex visual stimuli. Any observed classification performance likely reflects incidental patterns in the data rather than targeted neural specialization.
https://en.wikipedia.org/wiki/Postcentral_gyrus

VENTRAL TEMPORAL



The ventral temporal cortex is a critical region for high-level visual processing, including object and face recognition. It contains specialized areas such as the fusiform face area (FFA), which is highly responsive to facial stimuli.
https://en.wikipedia.org/wiki/Temporal_lobe

Note: Analysis and explanation of the code for each cell has been included in markdown format within the code itself. Screenshots are attached below.

Load the Dataset

```python
from nilearn import datasets
subj2 = datasets.fetch_haxby(subjects=[2])
```

Python

Analyse the Dataset

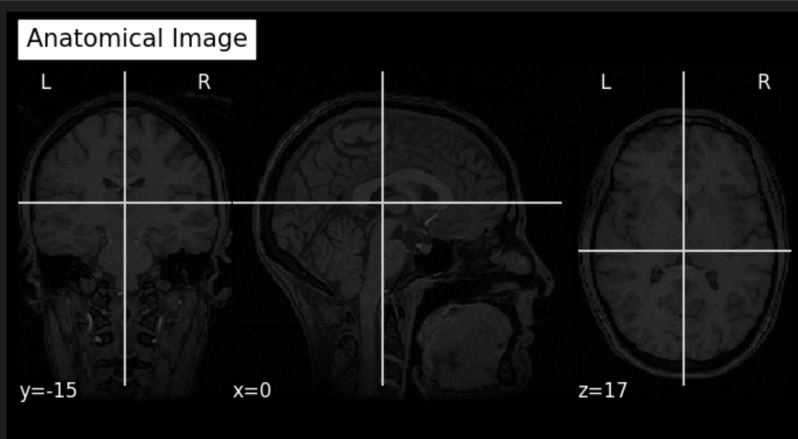
```python
print(subj2.keys())
```

Python

```
keys(['anat', 'func', 'session_target', 'mask_vt', 'mask_face', 'mask_house', 'mask_face_little', 'mask_house_little', 'mask', 'description'])
```

```python
from nilearn import plotting

# Visualize the anatomical image
plotting.plot_anat(subj2['anat'][0], title="Anatomical Image")
plotting.show()
```

[4] Python



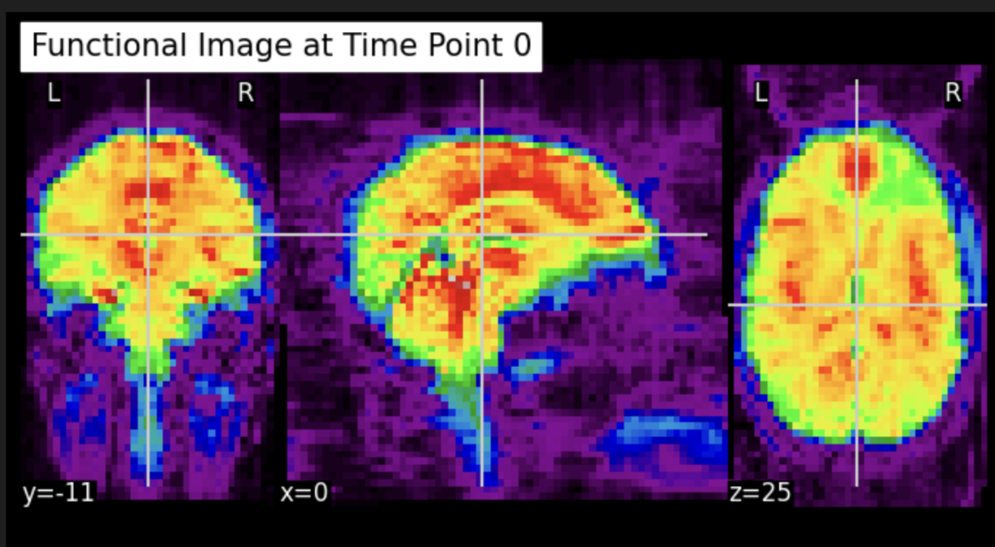
```python
from nilearn.image import load_img

# Load the first functional scan
func_img = load_img(subj2['func'][0])

# Print basic information
print(f"Functional Image Shape: {func_img.shape}")
```

[8]

```
Functional Image Shape: (40, 64, 64, 1452)
```

```python
from nilearn import plotting
from nilearn.image import index_img

# Extract the functional image at time point 0
func_img_t0 = index_img(func_img, 0)

# Plot the first time point
plotting.plot_epi(func_img_t0, title="Functional Image at Time Point 0")
plotting.show()
```

[9]

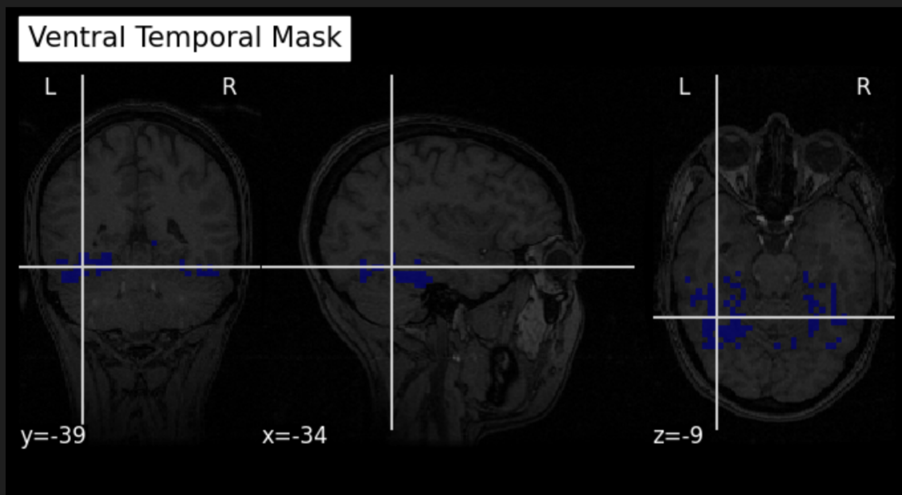


This isolates the ventral temporal region, which is a focus of our assignment.

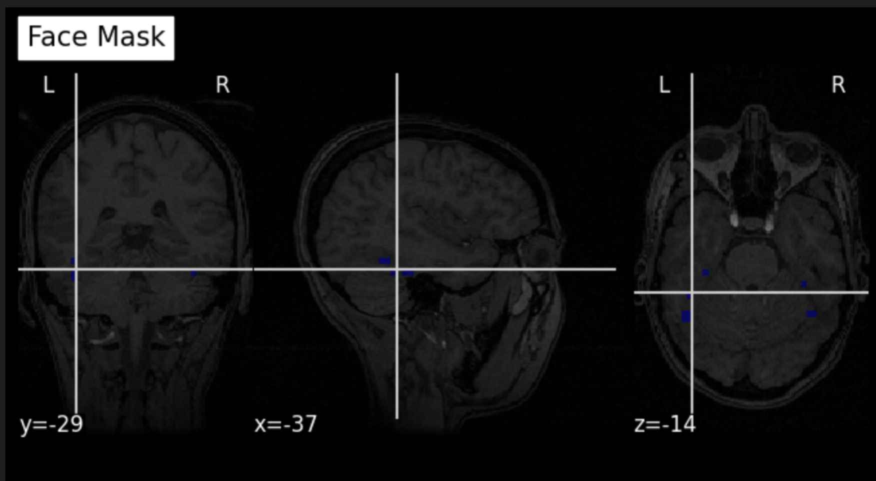
```
# Load and plot the mask
vt_mask_img = load_img(subj2['mask_vt'])
plotting.plot_roi(vt_mask_img, bg_img=subj2['anat'][0], title="Ventral Temporal Mask")
plotting.show()
```

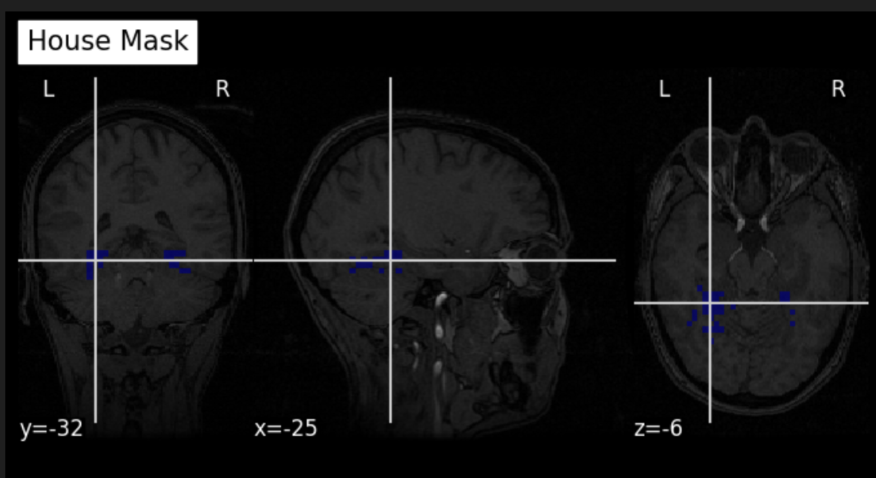
[10]

...



```
face_mask = load_img(subj2['mask_face'])
house_mask = load_img(subj2['mask_house'])

# Plot the masks
plotting.plot_roi(face_mask, bg_img=subj2['anat'][0], title="Face Mask")
plotting.plot_roi(house_mask, bg_img=subj2['anat'][0], title="House Mask")
plotting.show()
```





To understand how the ventral temporal region responds, extract time-series data corresponding to the voxels in the ventral temporal region .

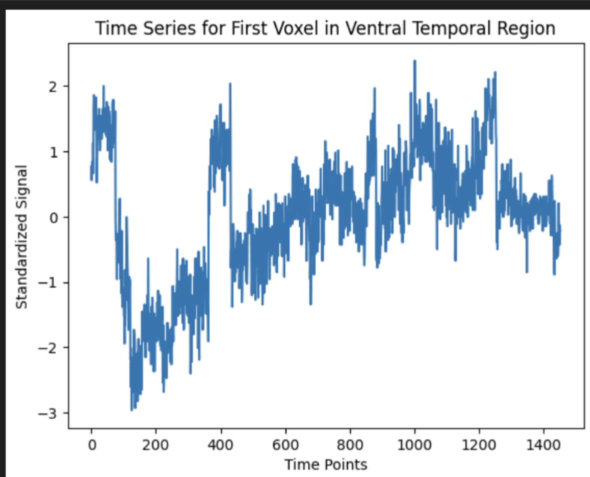
```
from nilearn.maskers import NiftiMasker

# Create a masker for the ventral temporal region
masker = NiftiMasker(mask_img=vt_mask_img, standardize=True)
time_series = masker.fit_transform(func_img)

# Inspect the extracted time series
print(f"Time-series shape: {time_series.shape}")
```

```
Time-series shape: (1452, 464)
```

Interpretation of Shape:

- Rows (1452): Time points, indicating how the brain's activity evolves over time.
- Columns (464): Spatial features (voxels) in the ROI, where activity is recorded.

```
import matplotlib.pyplot as plt

# Plot the first voxel's time series, like for each time series we just plot the first poxel
plt.plot(time_series[:, 0])
plt.title("Time Series for First Voxel in Ventral Temporal Region")
plt.xlabel("Time Points")
plt.ylabel("Standardized Signal")
plt.show()
```



NiftiMasker is a tool in Nilearn that simplifies preprocessing of fMRI data.

- `mask_img=subj2['mask_vt'][0]`: Specifies the ventral temporal (VT) mask, which defines the region of the brain from which time-series data will be extracted.
- `standardize="zscore_sample"`: Normalizes the time-series data for each voxel using z-score standardization. Each voxel's mean is subtracted, and the signal is scaled by its standard deviation. This ensures all signals are on a comparable scale.
- `detrend=True`: Removes low-frequency trends from the data, which could result from scanner drift or other artifacts. It helps focus on task-related signals.
- `high_variance_confounds=True`: Identifies and removes signals from voxels with high variance, which often correspond to noise or artifacts.

The masker contains these parameters, and after applying `fit_transform`, it will return the time-series data of the ventral temporal region. The `fit_transform` method applies the mask and preprocessing steps to the data, returning a 2D numpy array of the preprocessed time-series data.

In summary:

- NiftiMasker preprocesses neuroimaging data.
- `fit_transform` applies the mask and preprocessing steps to the data.
- Returns a 2D numpy array of the preprocessed time-series data.

```python
from nilearn.maskers import NiftiMasker
masker = NiftiMasker(mask_img=subj2['mask_vt'][0],
                     standardize="zscore_sample",
                     detrend=True,
                     high_variance_confounds=True)


time_series = masker.fit_transform(subj2['func'][0])
```

Python

```python
print(subj2['session_target'])
print(subj2['session_target'][0])
```

```
['/Users/prisha/nilearn_data/haxby2001/subj2/labels.txt']
/Users/prisha/nilearn_data/haxby2001/subj2/labels.txt
```

```python
import pandas as pd

# Load the behavioral data
behavioral = pd.read_csv(subj2['session_target'][0], sep=" ")

# Print the last few rows of the data
print(behavioral.tail())

# Extract and print unique labels
unique_labels = behavioral['labels'].unique()
print(f"Unique labels: {unique_labels}")
print(f"Number of unique labels: {len(unique_labels)}")
```

```
     labels  chunks
1447   rest      11
1448   rest      11
1449   rest      11
1450   rest      11
1451   rest      11
Unique labels: ['rest' 'scissors' 'face' 'cat' 'shoe' 'house' 'scrambledpix' 'bottle'
 'chair']
Number of unique labels: 9
```

```python
# Analyzing the shape that we would be working with
print(time_series.shape)
print(behavioral.shape)
```

```
(1452, 464)
(1452, 2)
```

Prepare the dataset that would be used to train the model for classification, conditioned on the labels face and house.

```python
import numpy as np

# Restrict to face, house conditions here
conditions = behavioral["labels"]
mask = conditions.isin(["face", "house"])

# Filter the time series and conditions using the mask
X = time_series[mask]
Y = conditions[mask]

# Print the shapes
print(time_series.shape)
print(X.shape)
print(Y.shape)
```

```
(1452, 464)
(216, 464)
(216,)
```

Using Different Models for Classification

Logistic Regression

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, random_state=42)

# Initialize and train Logistic Regression model
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)

# Predict and calculate accuracy
predicted_log_reg = log_reg.predict(X_test)
print("Logistic Regression Accuracy:", accuracy_score(y_test, predicted_log_reg))
```

```
Logistic Regression Accuracy: 0.9861111111111112
```

Random Forest Classifier

```python
from sklearn.ensemble import RandomForestClassifier

# Initialize and train Random Forest model
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train, y_train)

# Predict and calculate accuracy
predicted_rf = rf.predict(X_test)
print("Random Forest Accuracy:", accuracy_score(y_test, predicted_rf))
```

```
Random Forest Accuracy: 0.9861111111111112
```

```python
from sklearn.neighbors import KNeighborsClassifier

# Initialize and train K-Nearest Neighbors model
knn = KNeighborsClassifier(n_neighbors=10)
knn.fit(X_train, y_train)

# Predict and calculate accuracy
predicted_knn = knn.predict(X_test)
print("K-Nearest Neighbors Accuracy:", accuracy_score(y_test, predicted_knn))
```

```
K-Nearest Neighbors Accuracy: 0.9444444444444444
```

```python
from sklearn.svm import LinearSVC
svc=LinearSVC()

svc.fit(X_train, y_train)
predicted_svc = svc.predict(X_test)

print("LinearSVC Accuracy:", accuracy_score(y_test, predicted_svc))
```

```
LinearSVC Accuracy: 1.0
```

Out of all the Models that were tested (KNN, Random Forest,Logistic Regression), we see that LinearSVC performed the best because LinearSVC performed the best because it is particularly effective for high-dimensional datasets, such as our fMRI data, where the number of features (voxels) is much larger than the number of samples (time points). Additionally, the linear nature of the model makes it computationally efficient and well-suited for binary classification tasks like ours.

LeaveOneGroupOut cross validation strategy with LinearSVC

```python
accuracy_values1 = []

# Iterate through each of the 12 chunks (cross-validation)
for val in range(12):
    # Define the training and testing conditions for each chunk
    svc = LinearSVC()
    condition_mask_train = (mask) & (behavioral["chunks"] != val)
    condition_mask_test = (mask) & (behavioral["chunks"] == val)

    # Split the time series and labels based on the condition mask for training and testing
    X_train_selected = time_series[condition_mask_train]
    X_test_selected = time_series[condition_mask_test]
    y_train_selected = conditions[condition_mask_train]
    y_test_selected = conditions[condition_mask_test]

    # Check shapes of training and test sets for debugging
    print(f"\nTrain set shape (chunk {val}): {X_train_selected.shape}")
    print(f"Test set shape (chunk {val}): {X_test_selected.shape}")


    # Train the model on the training data
    svc.fit(X_train_selected, y_train_selected)

    # Predict the labels for the test data
    predicted_selected = svc.predict(X_test_selected)

    # Calculate accuracy for this chunk and append to the list
    accuracy = accuracy_score(y_test_selected, predicted_selected)
    accuracy_values1.append(accuracy)

    # Print the accuracy for the current chunk
    print(f"Accuracy for the test chunk {val}: {accuracy}")
```

Above is the code with corresponding markdown explanations. The next section presents the results.

VENTRAL TEMPORAL ALL FEATURES

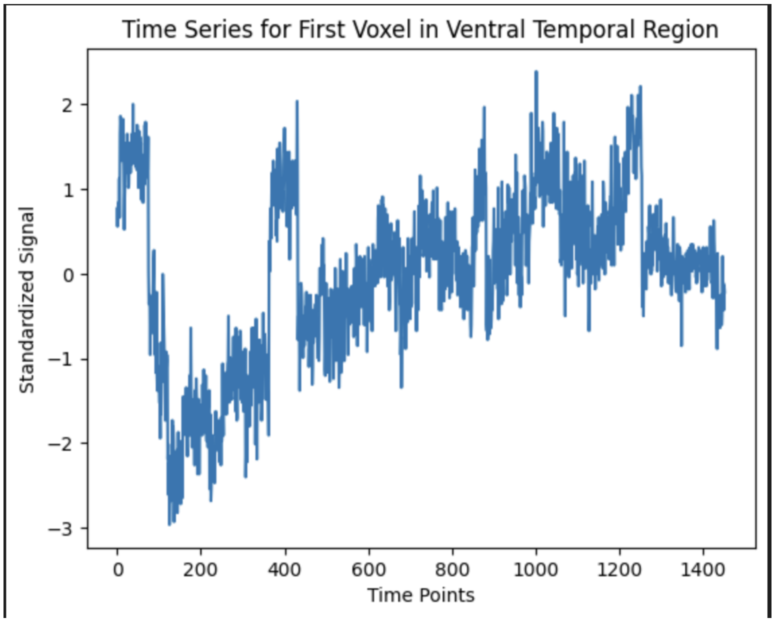
In analyzing the ventral temporal region of the brain, we used all available features.

Time-series shape: (1452, 464)

Interpretation of Shape:

- Rows (1452): Time points, indicating how the brain's activity evolves over time.
- Columns (464): Spatial features (voxels) in the ROI, where activity is recorded.

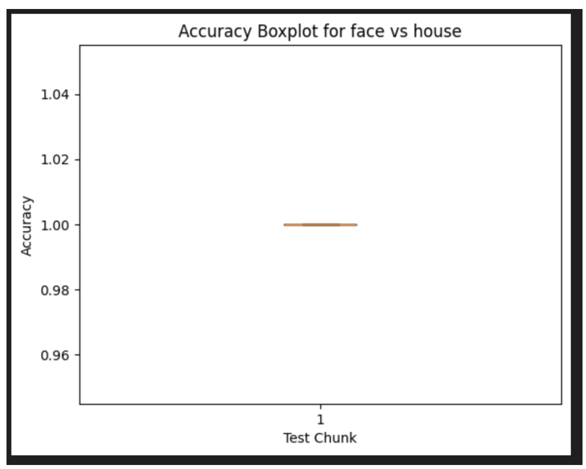
Labels were binary values (house or face) corresponding to the time series data.



FACE VS HOUSE :

MODEL USED : LINEAR SVC

```
Accuracy for the test chunk 0: 1.0
Accuracy for the test chunk 1: 1.0
Accuracy for the test chunk 2: 1.0
Accuracy for the test chunk 3: 1.0
Accuracy for the test chunk 4: 1.0
Accuracy for the test chunk 5: 1.0
Accuracy for the test chunk 6: 1.0
Accuracy for the test chunk 7: 1.0
Accuracy for the test chunk 8: 1.0
Accuracy for the test chunk 9: 1.0
Accuracy for the test chunk 10: 1.0
Accuracy for the test chunk 11: 1.0
```



Explanation for Perfect Accuracy in Face vs House Classification

Using the ventral temporal region of interest (ROI) with a LinearSVC model, the classification accuracy between faces and houses for subject 2 reached 1.0 across all test sets. This perfect score demonstrates the model's exceptional ability to distinguish between these two categories.

Several key factors likely contributed to this result:

1. **Distinctive Features**: The ventral temporal region contains clearly distinguishable patterns that the LinearSVC model can easily separate.
2. **Data Quality**: Subject 2's data shows clear, consistent patterns that enable effective learning and generalization.
3. **Model Performance**: The LinearSVC model is particularly well-suited for this binary classification task and this specific ROI.

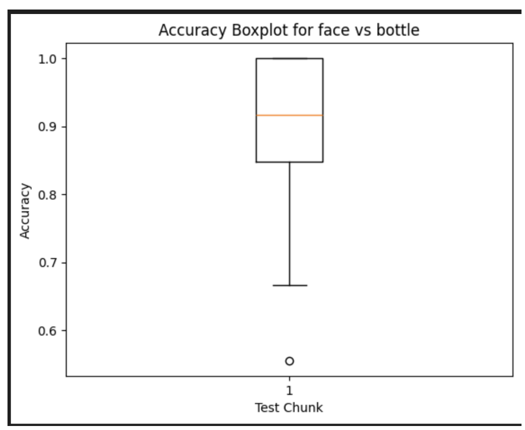
The ventral temporal cortex is known for its specialized role in face processing, particularly in the fusiform face area.
https://en.wikipedia.org/wiki/Fusiform_face_area

These results confirm that the ventral temporal ROI provides highly reliable information for discriminating between faces and houses in this subject's brain activity.

FACE VS BOTTLE :

MODEL USED : LINEAR SVC

```
Accuracy for test chunk 0: 0.6666666666666666
Accuracy for test chunk 1: 1.0
Accuracy for test chunk 2: 0.7222222222222222
Accuracy for test chunk 3: 0.5555555555555556
Accuracy for test chunk 4: 1.0
Accuracy for test chunk 5: 1.0
Accuracy for test chunk 6: 0.9444444444444444
Accuracy for test chunk 7: 1.0
Accuracy for test chunk 8: 0.8888888888888888
Accuracy for test chunk 9: 0.8888888888888888
Accuracy for test chunk 10: 0.8888888888888888
Accuracy for test chunk 11: 0.9444444444444444
```



Face vs. Bottle Classification with Linear SVC Model for Subject 2

For face vs. bottle classification using the Linear SVC model with Subject 2, we observed varying accuracy levels, with a mean of 87.5% and a standard deviation of 0.14. This performance indicates that distinguishing between faces and bottles is more challenging than face vs. house classification. Several factors contribute to this:

1. **Feature Overlap**: Bottles and faces share more visual properties (such as symmetry and contours) than houses do with faces, making the discrimination task more difficult.
2. **Data Variability**: The ventral temporal region shows more varied response patterns when processing faces and bottles compared to faces and houses.
3. **Model Sensitivity**: The Linear SVC model's sensitivity to subtle data differences results in varying accuracy across different data chunks.

While the model achieves good performance overall, the inherent complexity of distinguishing faces from bottles exceeds that of the face vs. house classification task.

VENTRAL TEMPORAL 50% FEATURES

We used np.random.choice to randomly select 50% of the features for training.

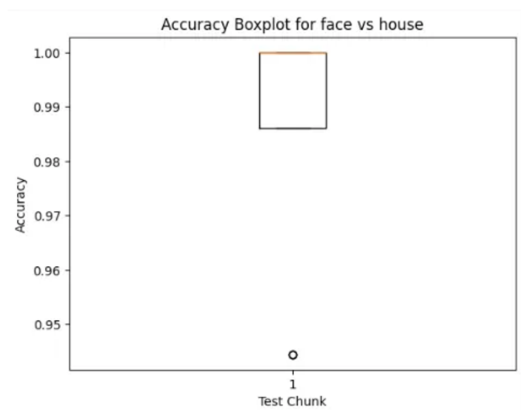
Original time series shape: (216, 464)
Shape with 50% random features: (216, 232)
(216, 232)
(216,)

```python
import numpy as np

# Restrict to face, house conditions here
conditions = behavioral["labels"]
mask = conditions.isin(["face", "house"])

# Filter the time series and conditions using the mask
X = time_series[mask]
Y = conditions[mask]

# Get the number of features (columns) in the time series
n_features = X.shape[1]

# Randomly select 50% of the features (columns)
random_feature_indices = np.random.choice(n_features, size=n_features // 2, replace=False)

# Select 50% random features
X_random = X[:, random_feature_indices]

# Print the shapes of the original and modified feature sets
print(f"Original time series shape: {X.shape}")
print(f"Shape with 50% random features: {X_random.shape}")

# Now X_random can be used for training and testing
X=X_random

print(X.shape)
print(Y.shape)
```

```
Original time series shape: (216, 464)
Shape with 50% random features: (216, 232)
(216, 232)
(216,)
```

FACE VS HOUSE :

```
Accuracy for the test chunk 0: 0.9444444444444444
Accuracy for the test chunk 1: 1.0
Accuracy for the test chunk 2: 1.0
Accuracy for the test chunk 3: 0.9444444444444444
Accuracy for the test chunk 4: 1.0
Accuracy for the test chunk 5: 1.0
Accuracy for the test chunk 6: 1.0
Accuracy for the test chunk 7: 1.0
Accuracy for the test chunk 8: 1.0
Accuracy for the test chunk 9: 1.0
Accuracy for the test chunk 10: 1.0
Accuracy for the test chunk 11: 0.9444444444444444
```



Analysis of Feature Reduction Impact on Face vs. House Classification

50% Random Features

- **Mean Accuracy:** 0.986
- **Standard Deviation:** 0.024

All Features

- **Accuracy:** 1.0

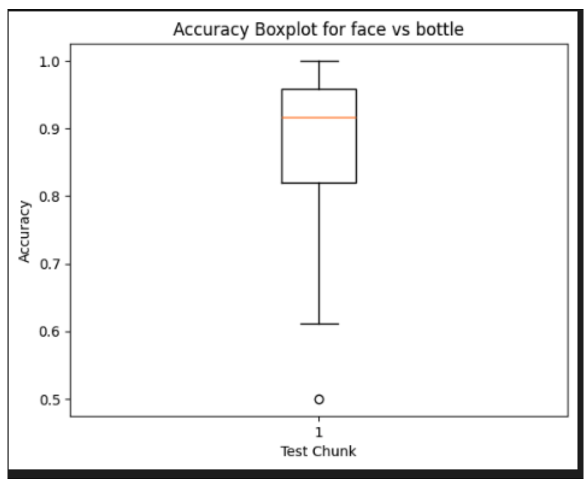
- **Standard Deviation:** 0

Key Findings

- Using fewer features leads to reduced classification accuracy, showing that complete feature access is essential for optimal performance.
- With all features, the model achieves perfect accuracy without variation, demonstrating exceptional stability.
- The reduced feature set introduces more variability, resulting in less reliable performance.
- Testing with different random feature selections yields inconsistent results, emphasizing the necessity of using the complete feature set.

FACE VS BOTTLE :

```
Accuracy for test chunk 0: 0.6111111111111112
Accuracy for test chunk 1: 0.9444444444444444
Accuracy for test chunk 2: 0.6111111111111112
Accuracy for test chunk 3: 0.5
Accuracy for test chunk 4: 1.0
Accuracy for test chunk 5: 0.9444444444444444
Accuracy for test chunk 6: 0.8888888888888888
Accuracy for test chunk 7: 0.8888888888888888
Accuracy for test chunk 8: 0.8888888888888888
Accuracy for test chunk 9: 1.0
Accuracy for test chunk 10: 1.0
Accuracy for test chunk 11: 0.9444444444444444
```



50% Features

- **Mean Accuracy:** 0.852
- **Standard Deviation:** 0.167

All Features

- **Mean Accuracy:** 0.875

- **Standard Deviation:** 0.142

Conclusion

Using 50% random features results in lower accuracy and higher standard deviation compared to using all features. This is expected, as fewer features provide less information for the model to learn from, leading to more variable results across different runs. This variation highlights the differing importance of individual voxels.

The face vs. bottle classification shows lower accuracy than face vs. house classification, both with full and 50% feature sets. This accuracy drop occurs because faces and bottles share more low-level visual features, making them harder to distinguish—especially when working with limited features.

POST CENTRAL WITH ALL FEATURES

Note that data was taken from the uploaded data link for subject 2's post-central region, using the feature.csv file and the code below. The shape of the data was

Time-series shape: (1452, 82), representing 1452 data points with 82 voxels as features

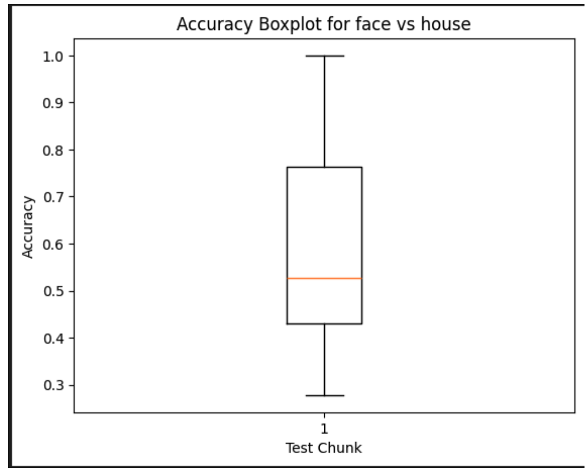
```
from nilearn.maskers import NiftiMasker
import pandas as pd

# take the time series from the data that is already provided
time_series = pd.read_csv('features.csv', header=None, sep=" ")

# Inspect the extracted time series
print(f"Time-series shape: {time_series.shape}")
```

[7] ✓ 0.0s

```
Time-series shape: (1452, 82)
```

FACE VS HOUSE

```
Accuracy for the test chunk 0: 0.5
Accuracy for the test chunk 1: 1.0
Accuracy for the test chunk 2: 0.9444444444444444
Accuracy for the test chunk 3: 0.4444444444444444
Accuracy for the test chunk 4: 0.3333333333333333
Accuracy for the test chunk 5: 0.5555555555555556
Accuracy for the test chunk 6: 0.5
Accuracy for the test chunk 7: 0.7222222222222222
Accuracy for the test chunk 8: 0.8888888888888888
Accuracy for the test chunk 9: 0.2777777777777778
Accuracy for the test chunk 10: 0.5555555555555556
Accuracy for the test chunk 11: 0.3888888888888889
```



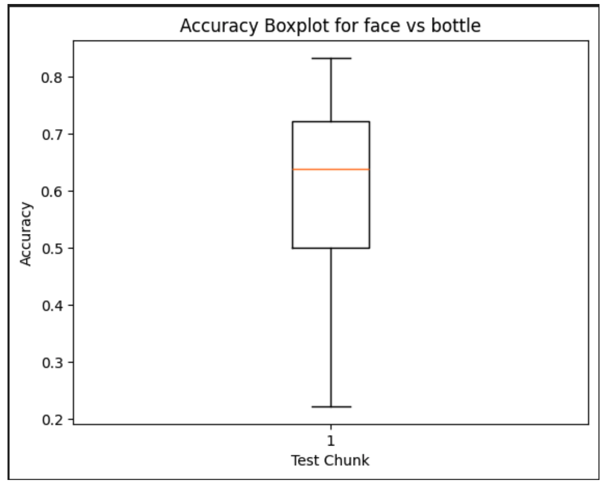
Analysis

- The post-central cortex is primarily involved in somatosensory processing and is not specialized for visual stimuli.
- The relatively low accuracy (59.26%) and high variability (standard deviation of 0.232) indicate limited and inconsistent discriminatory power for visual categories. This performance is notably weaker than the ventral temporal region with all features, which achieved 100% accuracy with zero standard deviation using Linear SVC. This difference exists because the ventral temporal region specializes in object detection, unlike the post-central region.

https://en.wikipedia.org/wiki/Postcentral_gyrus

FACE VS BOTTLE

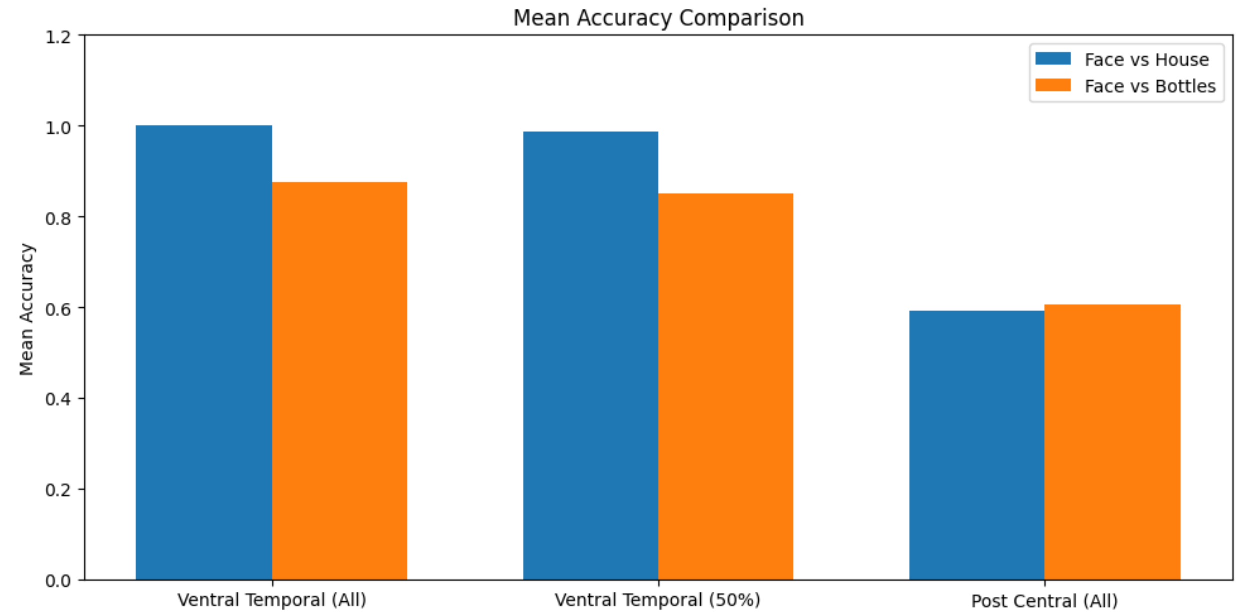
```
Accuracy for test chunk 0: 0.5
Accuracy for test chunk 1: 0.6111111111111112
Accuracy for test chunk 2: 0.8333333333333334
Accuracy for test chunk 3: 0.8333333333333334
Accuracy for test chunk 4: 0.7222222222222222
Accuracy for test chunk 5: 0.6666666666666666
Accuracy for test chunk 6: 0.2222222222222222
Accuracy for test chunk 7: 0.7222222222222222
Accuracy for test chunk 8: 0.6666666666666666
Accuracy for test chunk 9: 0.5
Accuracy for test chunk 10: 0.5555555555555556
Accuracy for test chunk 11: 0.4444444444444444
```

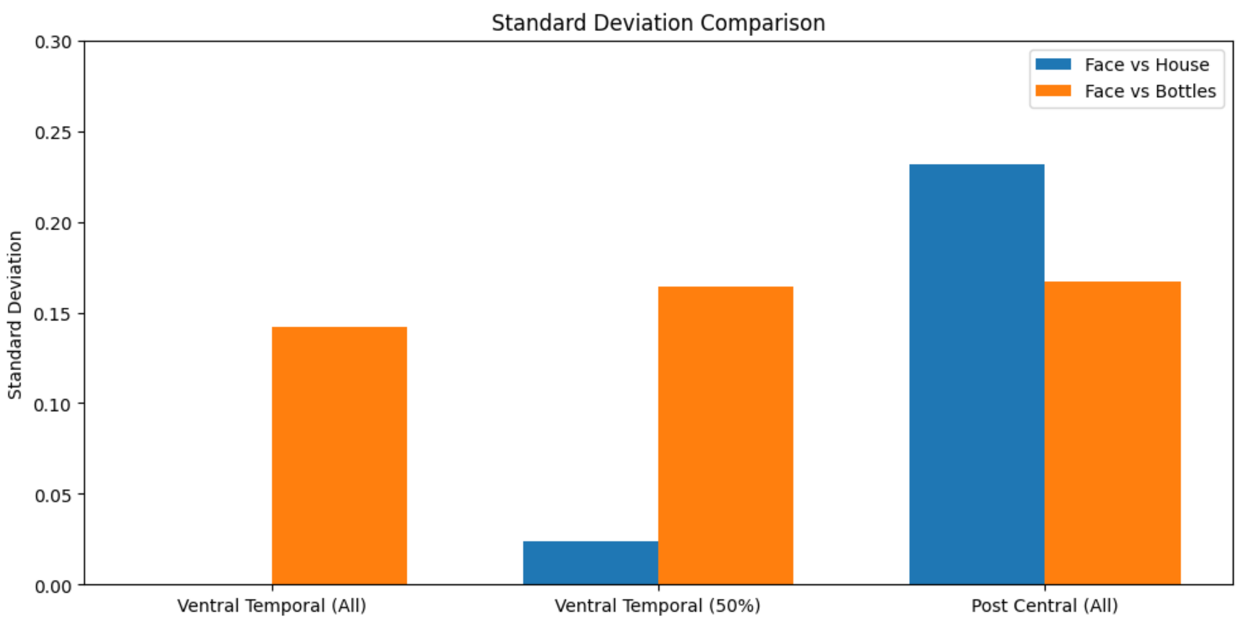


Mean Accuracy across all chunks: 0.61
Standard Deviation of Accuracy: 0.17

As expected, the accuracy for face vs. bottle classification is lower in the post-central region compared to the ventral temporal region. This is because the ventral temporal region specializes in object detection, while the post-central region does not. However, an interesting observation is that the face vs. bottle accuracy is slightly higher than the face vs. house accuracy in the post-central region.

OVERALL COMPARISION





Specialization of Brain Regions

- **Ventral Temporal Cortex**: Demonstrates higher accuracy and lower variability due to its role in visual object recognition.
- **Post-Central Cortex**: Shows poor performance with high variability as it is not specialized for visual tasks.

Effect of Feature Limitation

- **Impact on Accuracy**: Reducing features decreases accuracy and increases variability.
- **Importance of Complete Feature Sets**: Complete feature sets are crucial for robust classification.

Visual Category Comparisons

- **Face vs. Bottles**: Classification is challenging due to shared low-level visual features.
- **Face vs. House**: Classification is easier due to distinct structural differences.
- https://pubmed.ncbi.nlm.nih.gov/9151747/ : The fusiform face area: A module in human extrastriate cortex specialized for face perception.
- https://www.nature.com/articles/nrn3747 : The functional architecture of the ventral temporal cortex and its role in categorization.