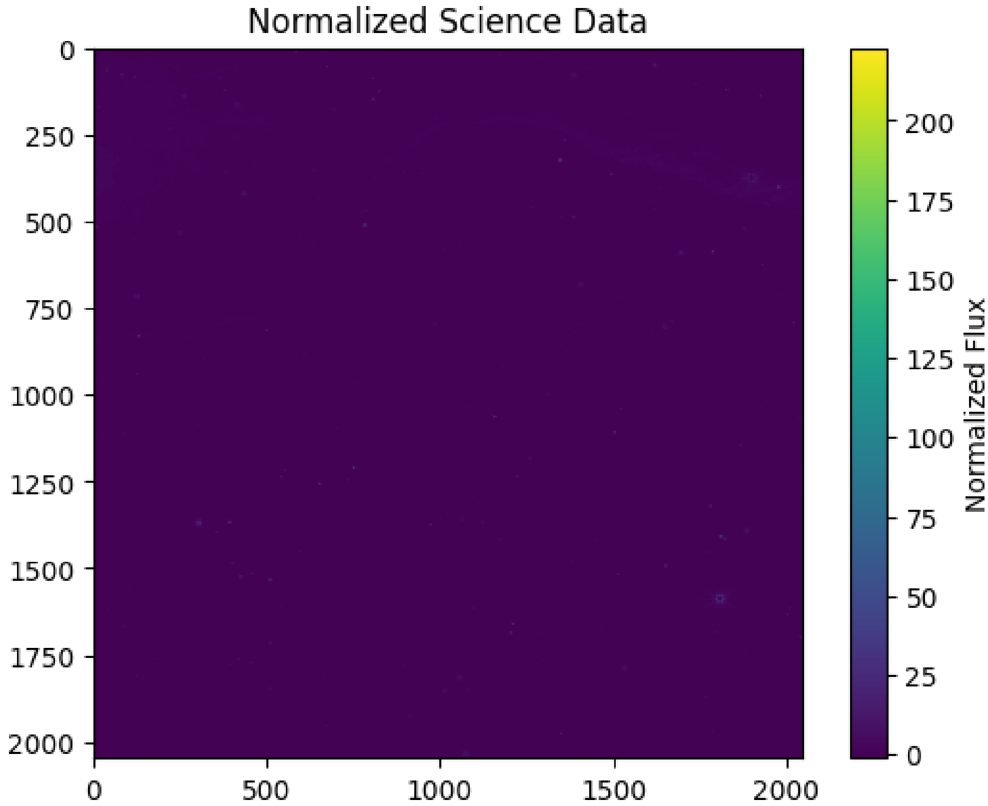
JWST

January 8, 2025

Loading and Preprocessing

[13]:
```
from astropy.io import fits
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Load FITS file
def load_fits(file_path):
    with fits.open(file_path) as hdul:
        sci_data = hdul['SCI'].data  # Science data
        dq_data = hdul['DQ'].data  # Data quality flags
        err_data = hdul['ERR'].data  # Uncertainty data
        header = hdul[0].header  # Metadata

    # Mask invalid data based on DQ flags (e.g., ignoring bad pixels)
    sci_data[dq_data != 0] = np.nan
    return sci_data, err_data, header

# Normalize the data (ignoring NaN values)
def normalize_data(data):
    flattened = data.flatten()
    valid_data = flattened[~np.isnan(flattened)]
    mean, std = np.mean(valid_data), np.std(valid_data)
    normalized = (data - mean) / std
    return np.nan_to_num(normalized)  # Replace NaNs with 0

# Process the FITS file
file_path = "JWST_data.fits"  # Replace with your FITS file
sci_data, err_data, header = load_fits(file_path)
sci_data_normalized = normalize_data(sci_data)

# Visualize the normalized data
plt.imshow(sci_data_normalized, cmap="viridis")
plt.colorbar(label="Normalized Flux")
```

```
plt.title("Normalized Science Data")
plt.show()
```



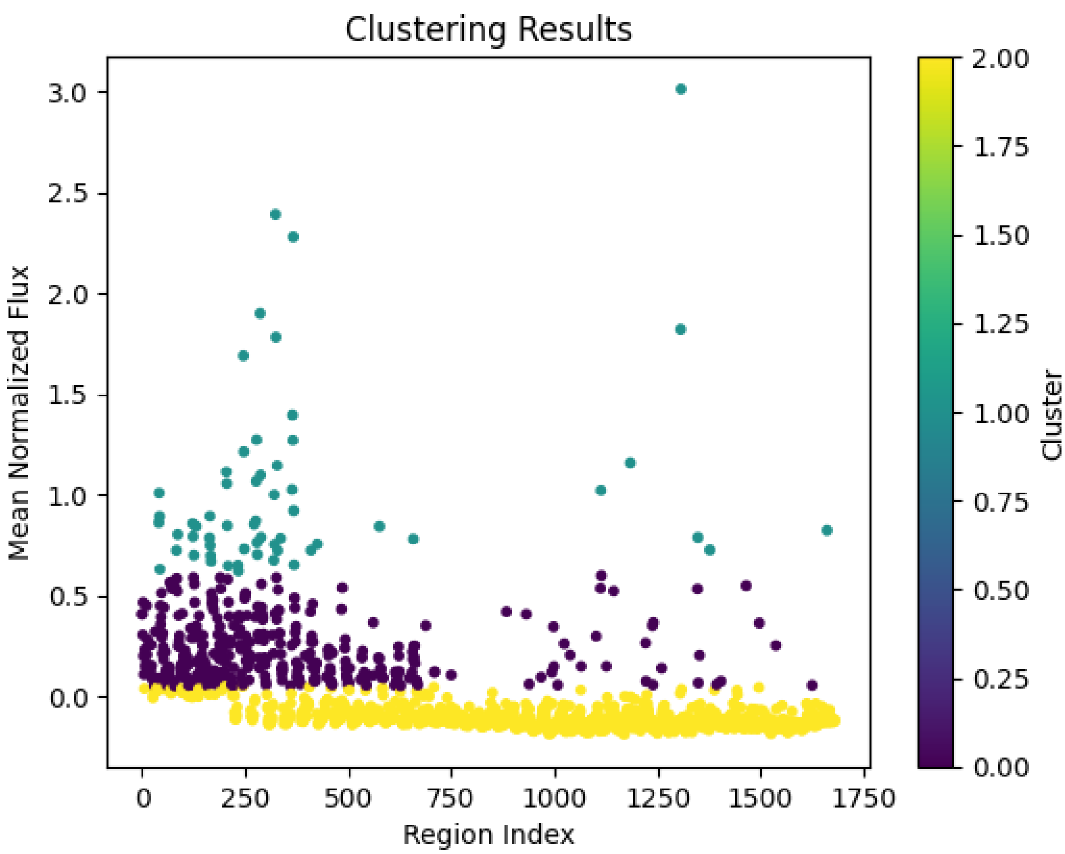
Extracting features for Clustering and RF

[14]:
```
def extract_features(data, region_size=50):
    """
    Divide the image into regions and compute mean flux per region.
    """
    rows, cols = data.shape
    features = []
    for i in range(0, rows, region_size):
        for j in range(0, cols, region_size):
            region = data[i:i + region_size, j:j + region_size]
            mean_flux = np.nanmean(region)  # Use nanmean to ignore NaNs
            features.append(mean_flux)
    return np.array(features).reshape(-1, 1)  # Reshape for sklearn models

features = extract_features(sci_data_normalized)
```

Clustering with K-means

```
[15]: # Perform k-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(features)

# Visualize the clusters
plt.scatter(range(len(features)), features, c=clusters, cmap="viridis", s=10)
plt.title("Clustering Results")
plt.xlabel("Region Index")
plt.ylabel("Mean Normalized Flux")
plt.colorbar(label="Cluster")
plt.show()
```

```
/home/chloy/.local/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

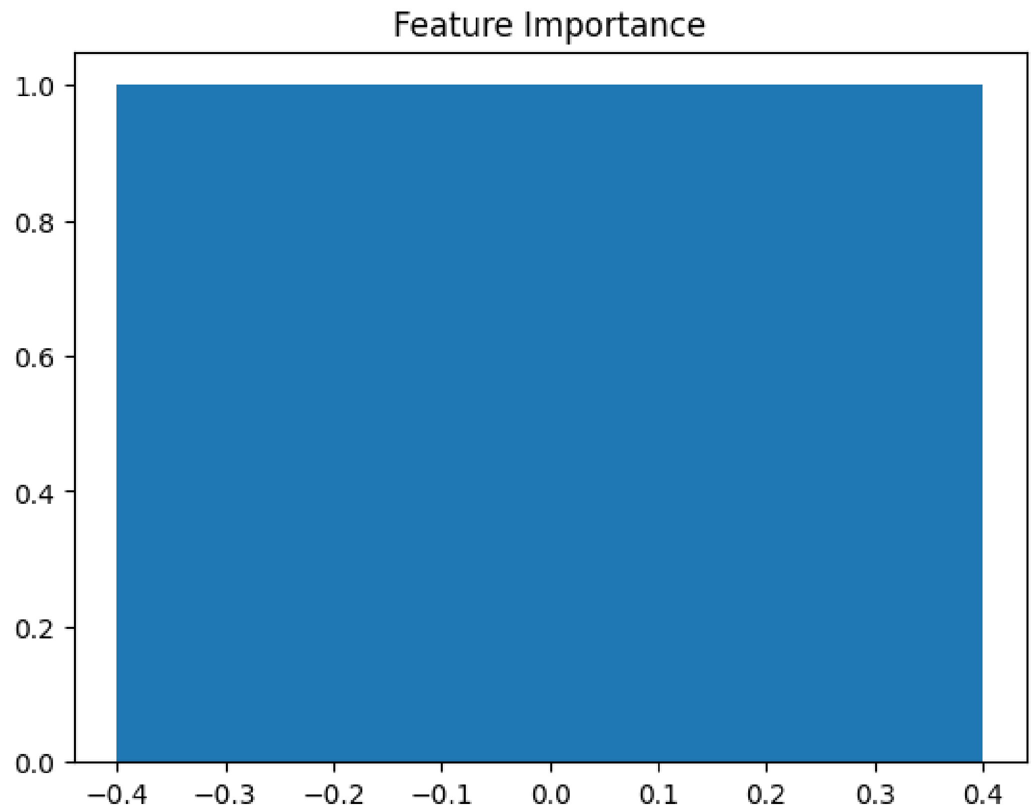


Random forest Regression

[16]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Generate synthetic labels for demonstration (replace with real labels)
labels = features.flatten() * 10 + np.random.normal(0, 0.5, len(features))  #␣
 ↪Example target values

# Split data
X_train, X_test, y_train, y_test = train_test_split(features, labels,␣
 ↪test_size=0.2, random_state=42)

# Train Random Forest
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predictions and evaluation
y_pred = rf.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Feature importance
importance = rf.feature_importances_
plt.bar(range(len(importance)), importance)
plt.title("Feature Importance")
plt.show()
```

```
Mean Squared Error: 0.9495418944157785
```


Feature Importance
1.0
0.8
0.6
0.4
0.2
0.0
−0.4
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4