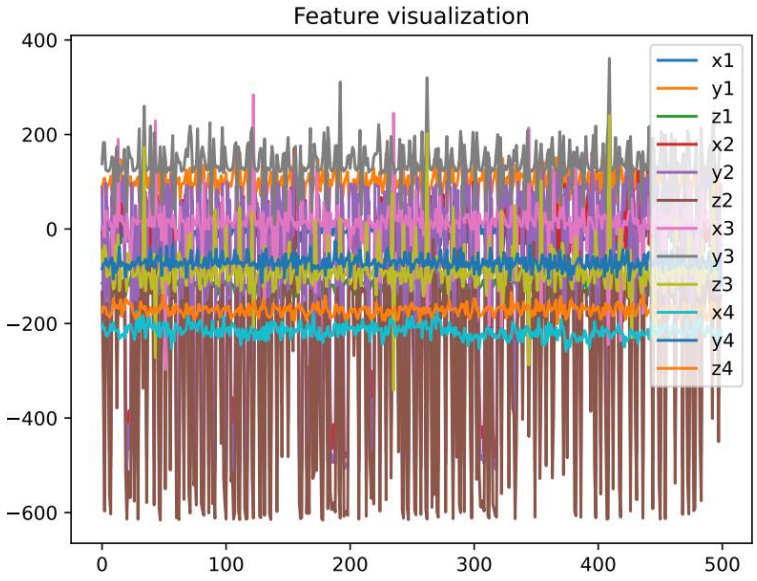
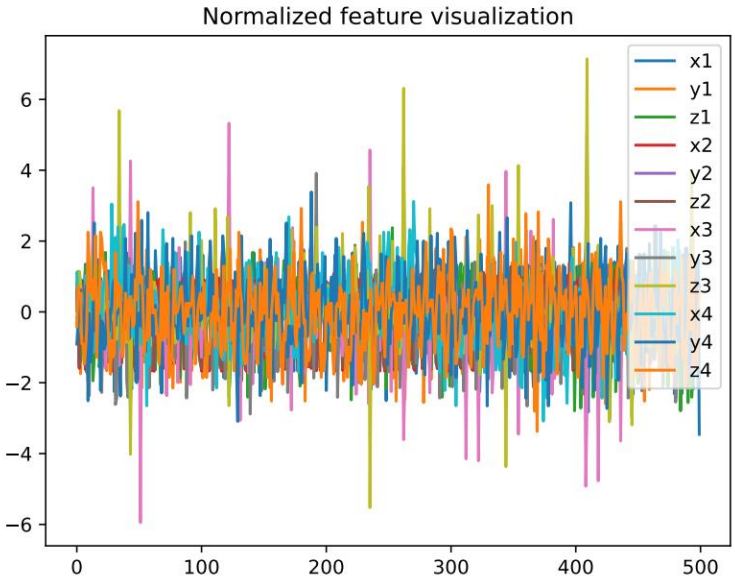
Lab 4 – Kay Burnham – ID 20220414

## 1




## 2




The data range before normalization was much wider than the normalized data. Normalization has also centered all data vertically about zero, allowing clearer analysis of relative peaks in data values.

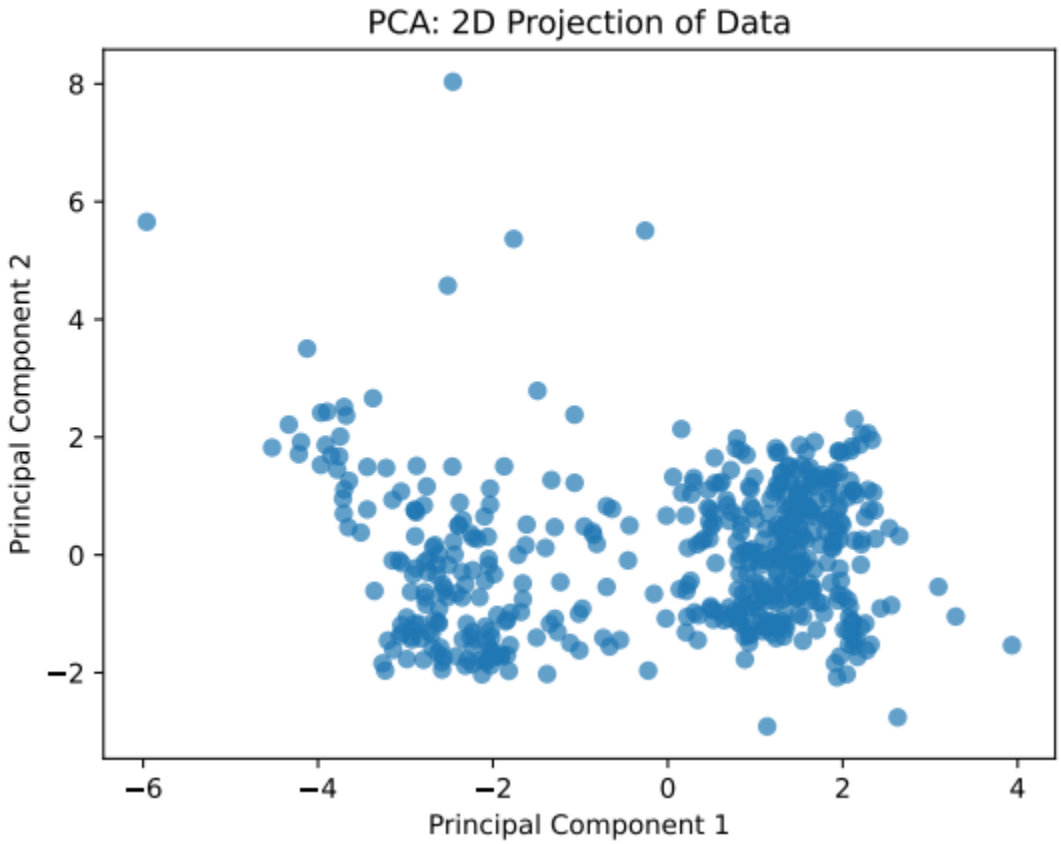
# 3

From lab4_PCA.py:

```
## Lab 4 code by Kay Burnham
# ID 20220414
# PCA: Question 1, corresponding to report part 1-6

import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
import os

dataset_path = os.path.join(os.path.dirname(__file__), 'dataset.csv')
dataset_full=pd.read_csv(dataset_path, delimiter=';')

# We want to take a subset of the dataset (the entire dataset is too large).

x = np.array(dataset_full.values[155750:156250, 6:18], dtype=float)
x_names=list(dataset_full.columns)[6:18]

print('feature shape: ', x.shape)
print('feature names: ', x_names)

# We can plot the 12 dimensions.
plt.figure()
t=[i for i in range(x.shape[0])]
for k in range(len(x_names)):
 plt.plot(t, x[:, k], label=x_names[k])

plt.legend(loc='upper right')
plt.title('Feature visualization')
plt.savefig('fig1.pdf')

# Calculate the mean and standard deviation:
m_x = np.mean(x, axis=0)
s_x = np.std(x, axis=0)

# Perform normalization
x_bar = (x - m_x) / s_x

plt.figure()
t=[i for i in range(x_bar.shape[0])]
for k in range(len(x_names)):
 plt.plot(t, x_bar[:, k], label=x_names[k])

plt.legend(loc='upper right')
plt.title('Normalized feature visualization')
plt.savefig('fig2.pdf')



def pca(x):
     # Step 1: Mean-center the data
        # Calculate the mean and standard deviation:
     m_x = np.mean(x, axis=0)
     s_x = np.std(x, axis=0)
```

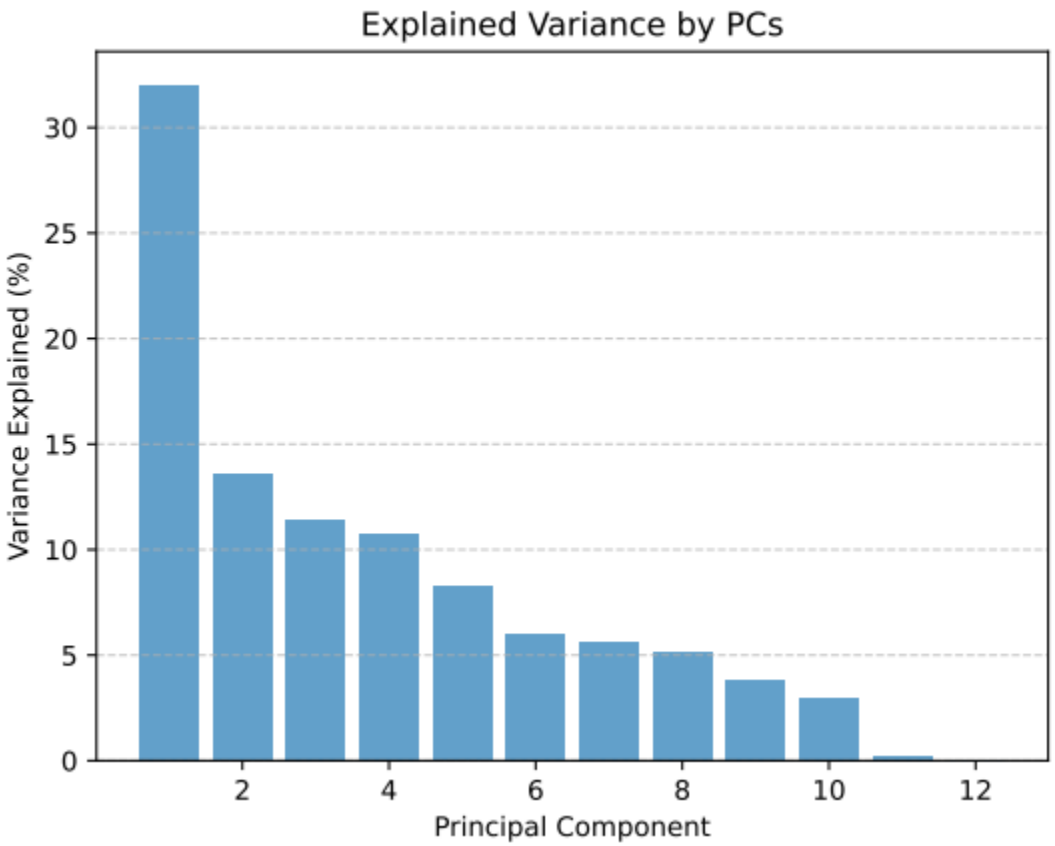
```
        # Perform normalization
    x_bar = (x - m_x) / s_x

    # Step 2: Compute the covariance matrix (use np.cov)
    x_cov = np.cov(x_bar, rowvar=False)

    # Step 3: Perform eigen decomposition (use np.linalg.eigh)
    x_eigval, x_eigvect = np.linalg.eigh(x_cov)

    # Step 4: Sort the eigenvalues AND eigenvectors in decreasing order
    # (Hint: use np.argsort on eigenvalues)
    idx = np.argsort(x_eigval)[::-1]
    x_eigval = x_eigval[idx]
    x_eigvect = x_eigvect[:, idx]

    # Step 5: Select the sorted eigenvectors as principal components (pcs)
    k = 2 #assuming 2 principal components (becomes 2D for plotting)
    pcs = x_eigvect[:, :k]

    # Step 6: Compute the variance explained by each principal component
(explained)
    explained = x_eigval / np.sum(x_eigval)

    # Step 7: Project the data onto principal component axes to obtain scores
    scores = np.dot(x_bar, pcs)

    return pcs, scores, explained


pcs, scores, explained = pca(x)

# Following plots are for parts 4, 5, and 6

plt.figure()
plt.scatter(scores[:, 0], scores[:, 1], alpha=0.7)
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA: 2D Projection of Data")
plt.savefig('fig4.pdf')

plt.figure()
plt.bar(range(1, len(explained) + 1), explained * 100, alpha=0.7)
plt.xlabel("Principal Component")
plt.ylabel("Variance Explained (%)")
plt.title("Explained Variance by PCs")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.savefig('fig5.pdf')

plt.figure()
cumulative_variance = np.cumsum(explained)
plt.bar(range(1, len(explained) + 1), cumulative_variance * 100, alpha=0.7)
plt.xlabel("Number of Principal Components")
plt.ylabel("Cumulative Variance Explained (%)")
plt.title("Cumulative Explained Variance")
plt.axhline(y=98, color='gray', linestyle='--', label="98% threshold")
plt.legend()
plt.grid(alpha=0.7)
plt.savefig('fig6.pdf')
```

## 4



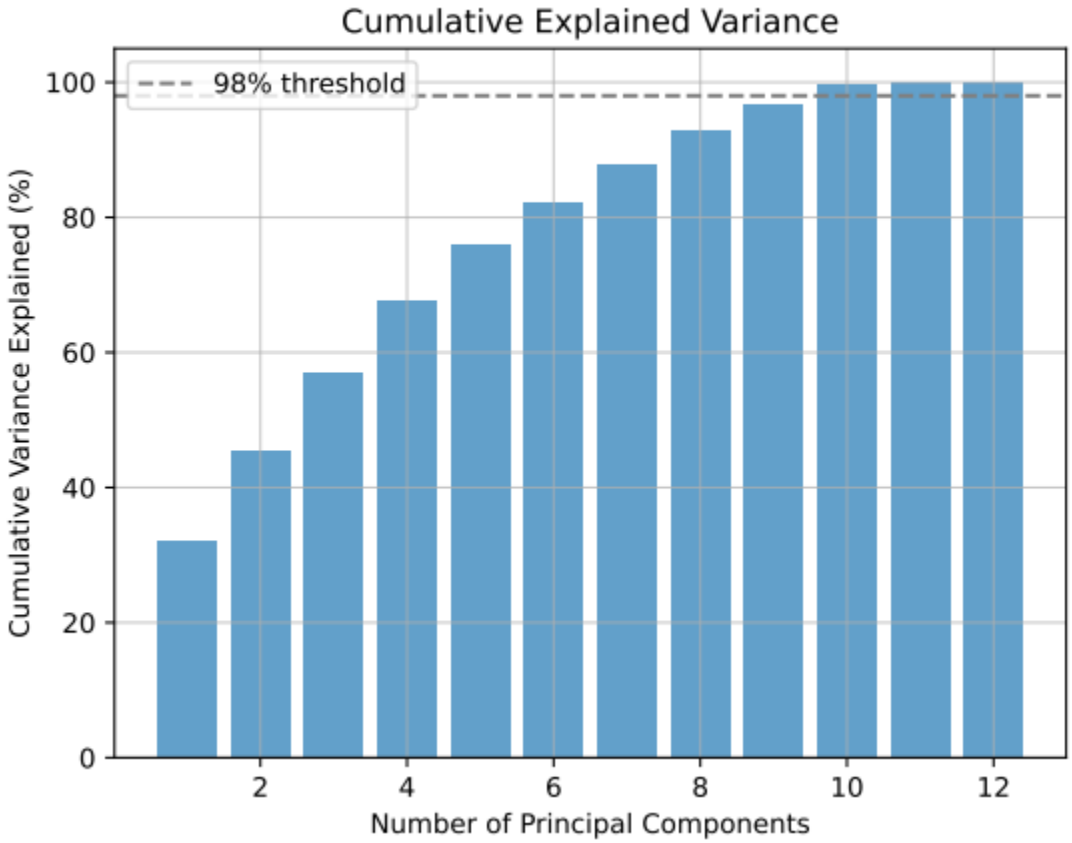

The first two principle components effectively split the data into two vague clusters, capturing the most important aspects of the original data set. These clusters are ‘vaguely negative’ or ‘vaguely positive’ along the x axis, and mostly contained within -2 to +2 on the y axis, though with less clear clustering than in the x direction.

5



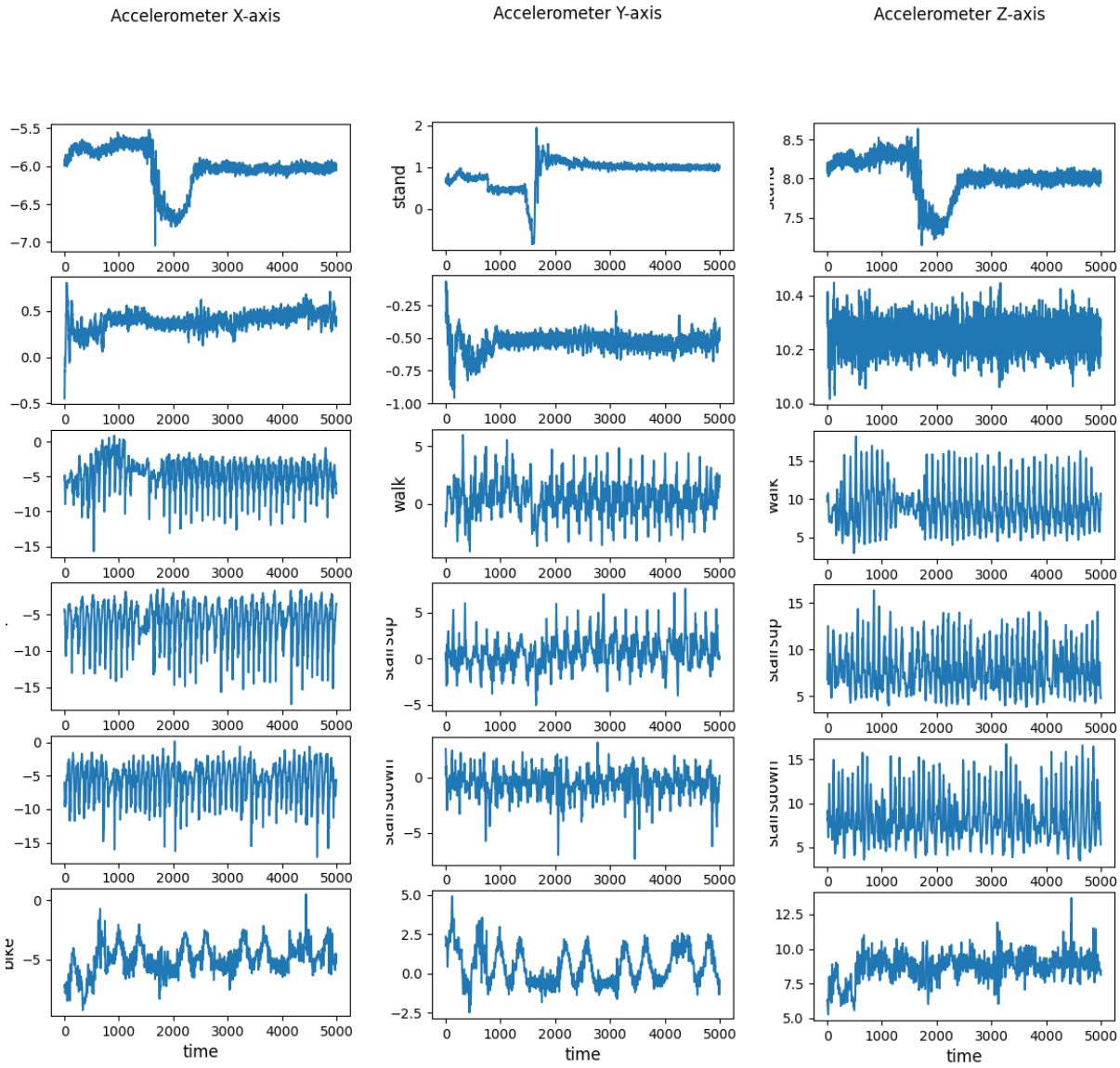

As shown above, the highest variance within the data set is seen in lower numbers of principal components, with a steep drop-off then gradual decline. This indicates that 2 PCs result in approx. 14% explained variance.

6




A minimum of 10 principal components are required to capture at least 98% of the information from the original dataset.

7


Accelerometer X-axis
Accelerometer Y-axis
Accelerometer Z-axis
stand
walk
time
time
time

## 8

Changes were made to the code to accomplish the following:

a) Modified function to extract additional features (max, min, mean, standard deviation, skewness, entropy)
b) Modified function call to create two sets of extracted features with two window sizes (500 vs 250 samples)

See code below:

```
## Lab 4 code by Kay Burnham
# ID 20220414
# Feature Extraction: Question 2, corresponding to report part 7

import pandas as pd
import numpy as np
import scipy as sp
import os

dataset_path = os.path.join(os.path.dirname(__file__),
'raw_accelerometer_dataset.csv')

def extract_features(filename,
                     segment_size):

    # Read raw data
    raw_data = pd.read_csv(filename, delimiter=',')

    # Initialize parameters
    feature_set = pd.DataFrame()
    event = 0
    classes = raw_data['Class'].unique()

    for task, class_name in enumerate(classes):
        # Filter data for current class/activity
        class_data = raw_data[raw_data['Class'] == class_name].iloc[:, 1:4]

        # Create group IDs
        num_samples = len(class_data)
        group_id = np.repeat(np.arange(event, event +
np.ceil(num_samples/segment_size)),
                             segment_size)[:num_samples]
        event += len(np.unique(group_id))

        class_data['group_id'] = group_id

        # Initialize feature containers
        features = []


        # Calculate features per segment
        for g_id, group in class_data.groupby('group_id'):
            data = group.iloc[:, :3]
```

```
            # Normalize data for entropy
            # Calculate the mean and standard deviation:
            m_data = np.mean(data, axis=0)
            s_data = np.std(data, axis=0)

            # Perform normalization
            norm_data = (data - m_data) / s_data

            feature_row = [
                *np.max(data, axis=0), # max_x, max_y, max_z
                *np.min(data, axis=0),
                *np.mean(data, axis=0),
                *np.std(data, axis=0),
                *sp.stats.skew(data, axis=0),
                *sp.stats.entropy(norm_data, axis=0)
            ]

            features.append(feature_row)

        # Create feature DataFrame for current class
        columns = [
            'max_x', 'max_y', 'max_z',
            'min_x', 'min_y', 'min_z',
            'mean_x', 'mean_y', 'mean_z',
            'std_x', 'std_y', 'std_z',
            'skew_x', 'skew_y', 'skew_z',
            'entropy_x', 'entropy_y', 'entropy_z',
        ]

        class_features = pd.DataFrame(features, columns=columns)
        class_features['activity'] = task

        feature_set = pd.concat([feature_set, class_features], ignore_index=True)

    # Save results
    feature_set.to_csv(f'features_{segment_size}.csv', index=False)
    return feature_set

if __name__=='__main__':

    # TODO: please update the code to extract additional features.
    extract_features(filename=dataset_path,
                     segment_size = 250)

    extract_features(filename=dataset_path,
                     segment_size = 500)
```