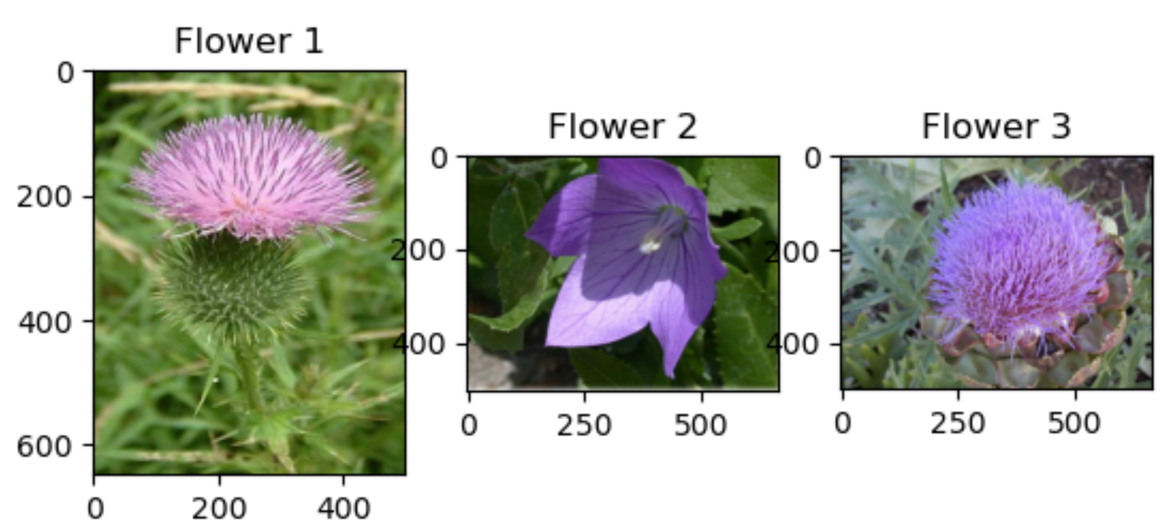
In [1]:
```python
# Import necessary libraries
import os
import time
import skdim
import numpy as np
import numpy as gfg
import pandas as pd
import seaborn as sns
import matplotlib.cm
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from PIL import Image
from sklearn.decomposition import PCA
from sklearn.decomposition import KernelPCA
from sklearn.manifold import SpectralEmbedding
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import classification_report, confusion_matrix
```

In [2]:
```python
# import the necessary packages
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

# create subplots to display multiple images
plt.subplot(1, 3, 1) # create a subplot with 1 row, 3 columns, and index position 1
image1 = mpimg.imread('/Users/preciousworgu/Downloads/DATA 604 Final Project/archive/dataset/train/14/image_06050.jpg') # read the first image
plt.title('Flower 1') # set the title of the first image
plt.imshow(image1) # show the first image

plt.subplot(1, 3, 2) # create a subplot with 1 row, 3 columns, and index position 2
image2 = mpimg.imread('/Users/preciousworgu/Downloads/DATA 604 Final Project/archive/dataset/train/19/image_06162.jpg') # read the second image
plt.title('Flower 2') # set the title of the second image
plt.imshow(image2) # show the second image

plt.subplot(1, 3, 3) # create a subplot with 1 row, 3 columns, and index position 3
image3 = mpimg.imread('/Users/preciousworgu/Downloads/DATA 604 Final Project/archive/dataset/train/29/image_04110.jpg') # read the third image
plt.title('Flower 3') # set the title of the third image
plt.imshow(image3) # show the third image
plt.show() # display the subplots
```



In [3]:
```python
# define a function that reads all the images in a folder and stores them as arrays
def flower_data(folder, flower):
    path = '/Users/preciousworgu/Downloads/DATA 604 Final Project/archive/dataset'
    path = os.path.join(path, folder, flower)
    files = [os.path.join(path, f) for f in os.listdir(path) if os.path.isfile(os.path.join(path, f))]
    imgs = []
    for file in files:
        img = Image.open(file)
        imageToMatrice = gfg.asarray(img)
        imageToMatrice = np.mean(imageToMatrice, axis=-1)
        imageToMatrice = imageToMatrice.reshape(-1)
        imgs.append(imageToMatrice)
    return imgs
```

In [4]:
```python
# define a function that creates a dataframe for images of a given flower type
def flower_df(folder1, folder2, flower, label):
    flower_lst1 = flower_data(folder1, flower)
    flower_lst2 = flower_data(folder2, flower)
    length = list(str(i) for i in range(10000))
    flower_df = pd.DataFrame()
    flower_df = flower_df.append(flower_lst1)
    flower_df = flower_df.append(flower_lst2)
    flower_df['label']= label
    return flower_df
```

In [5]:
```python
# create dataframes for images of three flower types with their respective labels
flower1_df = flower_df('train','valid', '14','0').sample(frac=1)
flower2_df = flower_df('train','valid', '19','1').sample(frac=1)
flower3_df = flower_df('train','valid', '29','2').sample(frac=1)
```

In [6]:
```
# Print the shape of each dataset
print(flower1_df.shape)
print(flower2_df.shape)
print(flower3_df.shape)
```

```
(45, 411001)
(42, 401501)
(69, 445501)
```

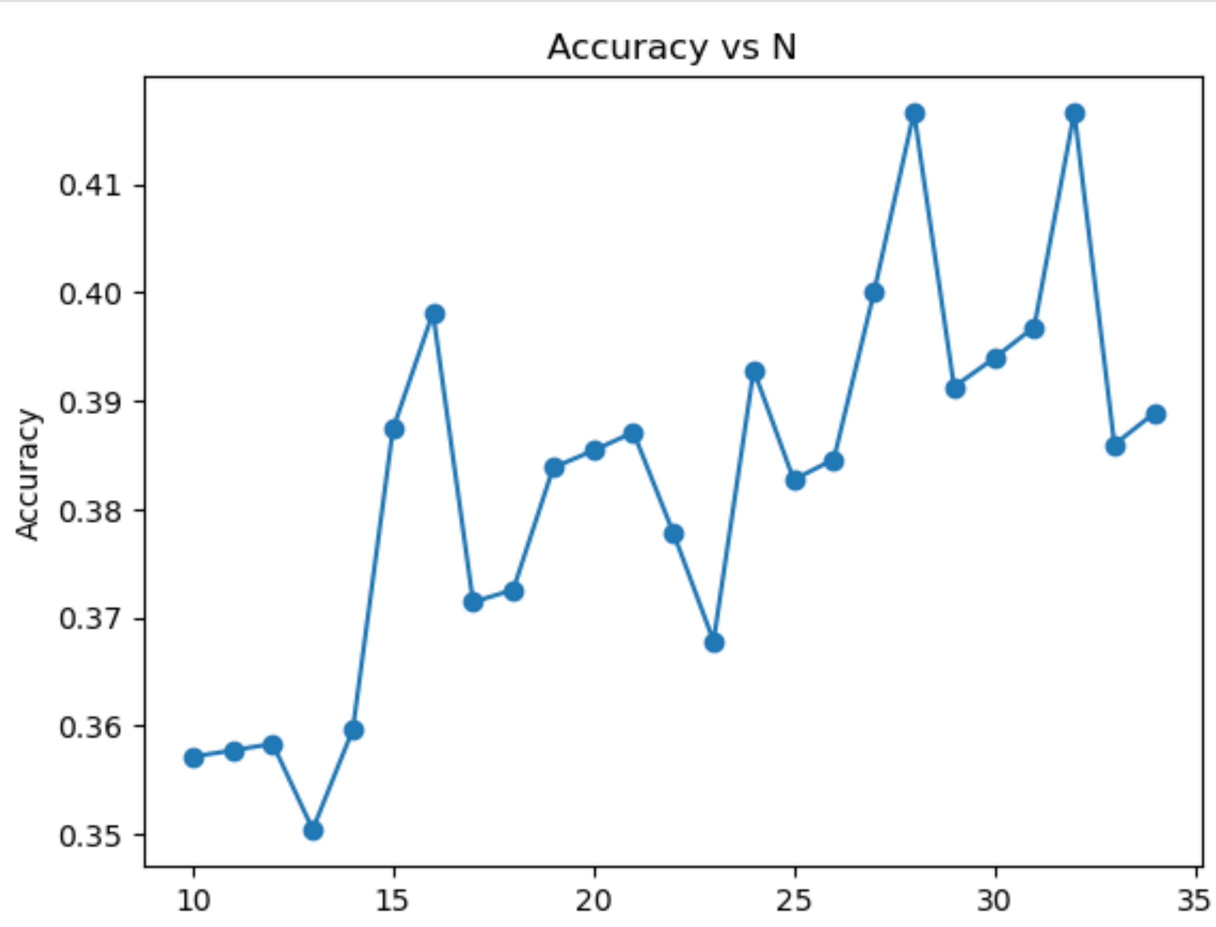
In [7]:
```
# Concatenate the three flower dataframes into a single dataframe using pd.concat
flowers_df = pd.concat([flower1_df, flower2_df, flower3_df], ignore_index=True)

# Fill any missing values in the concatenated dataframe with 0 using the inplace parameter
flowers_df.fillna(0, inplace=True)
```

In [8]:
```
# define a function that splits the dataframe into a training set and a test set,
def tts_by_class(df, n):
    train = pd.DataFrame()
    for i in range(3):
        train_oneclass = df.loc[df['label']==str(i)].iloc[:n]
        train = train.append(train_oneclass)
    test = df.loc[df.index.difference(train.index)]
    return train, test
```

In [9]:
```
# define a function that uses the KNN algorithm to classify test data
def knn_10(train, test, dist='euclidean'):
    X_train = train.drop(columns=['label'])
    y_train = train['label'].copy()
    X_test = test.drop(columns=['label'])
    y_test = test['label'].copy()
    neigh = KNeighborsClassifier(n_neighbors=10, metric=dist)
    neigh.fit(X_train, y_train)
    y_pred=neigh.predict(X_test)
    return y_pred
```

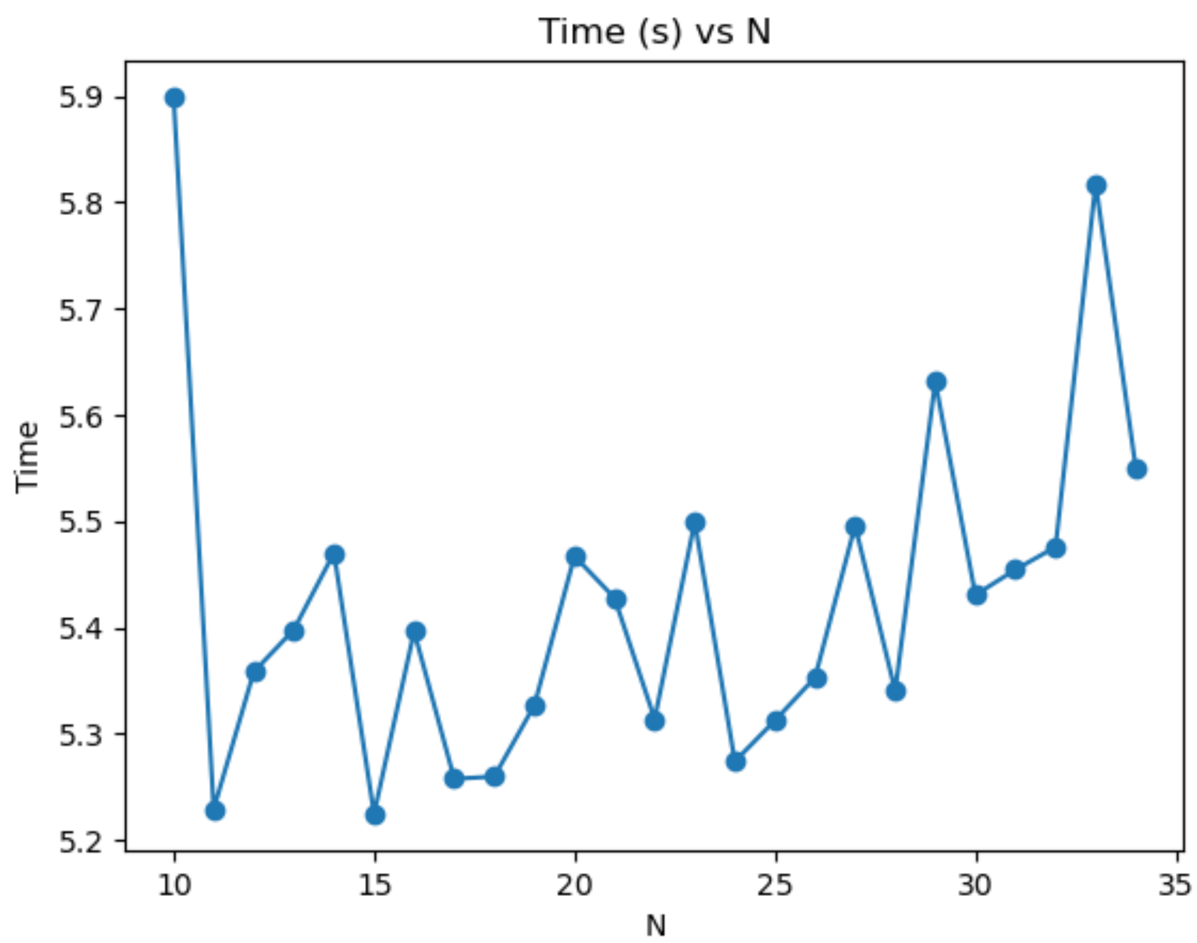
In [232…
```
# initialize empty lists to store the calculated execution times and accuracy-vs-size data
cal_time = []
acc_vs_size = []

# Define the range of training set
train_num = list(range(10,35,1))
for i in train_num:
    start_time = time.time()
    train_testn, test_testn = tts_by_class(flowers_df,i)
    predict_testn = knn_10(train_testn, test_testn)
    global_acc=sum(np.array(predict_testn)==np.array(test_testn['label']))/len(test_testn)
    end_time = time.time()
    acc_vs_size.append(global_acc)
    cal_time.append(end_time-start_time)
```

In [233…
```
plt.plot(train_num, acc_vs_size,'-o')
plt.xlabel("N")
plt.ylabel("Accuracy")
plt.title("Accuracy vs N")
plt.show()
```



In [235…

```
plt.plot(train_num, cal_time,'-o')
plt.xlabel("N")
plt.ylabel("Time")
plt.title("Time (s) vs N")
plt.show()
```



In [212…

```
flower1_train = flower1_df[:28]
flower1_val = flower1_df[28:35]
flower1_test = flower1_df[35:42]
flower2_train = flower2_df[:28]
flower2_val = flower2_df[28:35]
flower2_test = flower2_df[35:]
flower3_train = flower3_df[:28]
flower3_val = flower3_df[28:35]
flower3_test = flower3_df[35:42]
```

In [213…

```
train_df = pd.concat([flower1_train, flower2_train, flower3_train], ignore_index=True).sample(frac=1)
train_df.fillna(0, inplace=True)
val_df = pd.concat([flower1_val, flower2_val, flower3_val], ignore_index=True).sample(frac=1)
val_df.fillna(0, inplace=True)
test_df = pd.concat([flower1_test, flower2_test, flower3_test], ignore_index=True).sample(frac=1)
test_df.fillna(0, inplace=True)
```

In [214…

```
print(len(train_df))
print(len(val_df))
print(len(test_df))
```

```
84
21
21
```

In [215…

```
def feature_label_split(train, val, test):
    X_train = train.copy().drop(columns=['label'])
    y_train = train['label'].copy()
    X_val = val.copy().drop(columns=['label'])
    y_val = val['label'].copy()
    X_test = test.copy().drop(columns=['label'])
    y_test = test['label'].copy()
    return X_train, y_train, X_val, y_val, X_test, y_test
```

In [268…

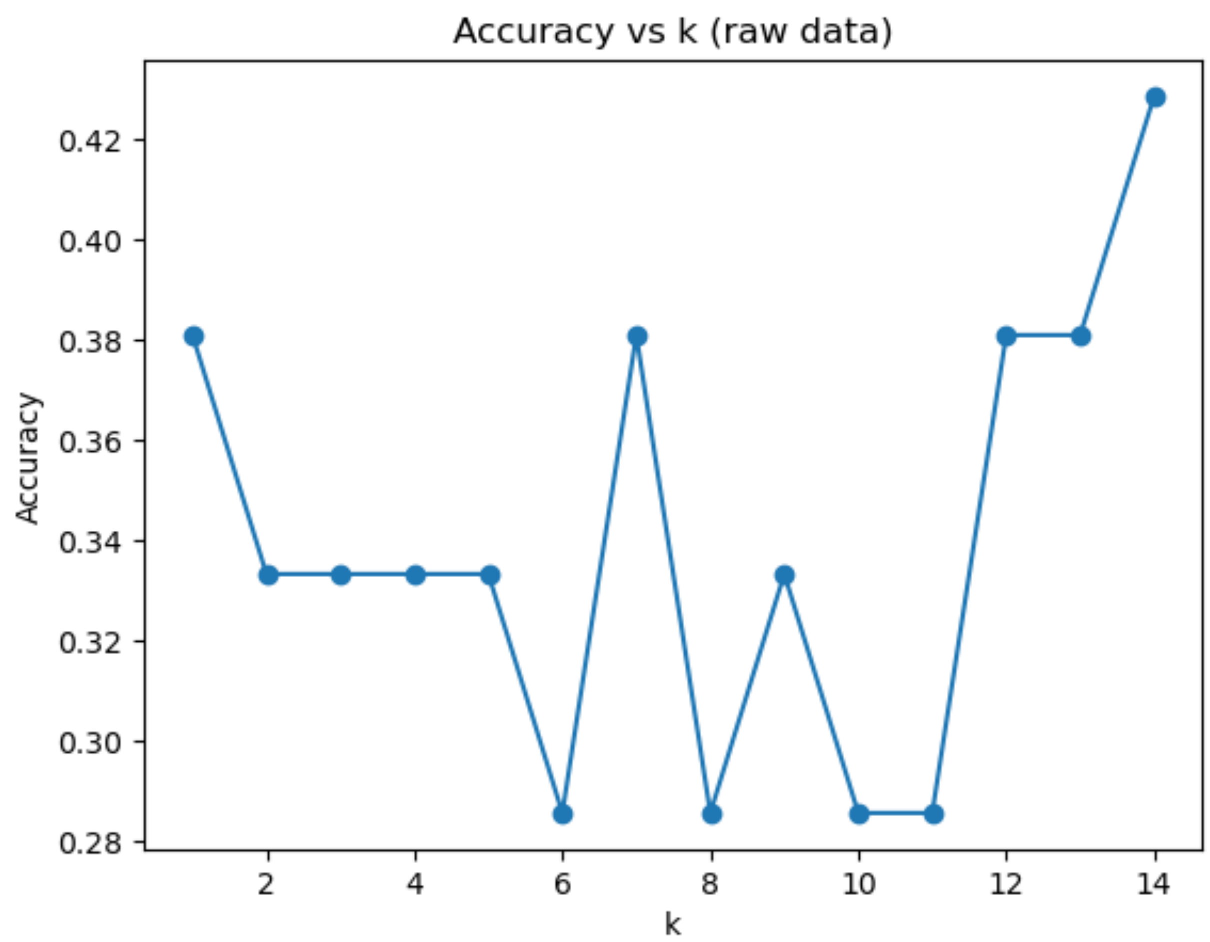
```
X_train, y_train, X_val, y_val, X_test, y_test = feature_label_split(train_df, val_df, test_df)
```

In [269…

```
def test_diff_k_acc(X_tr, y_tr, X_v, y_v):
    acc_lst = []
    for i in range(1,15):
        neigh = KNeighborsClassifier(n_neighbors=i)
        neigh.fit(X_tr, y_tr)
        y_pred = neigh.predict(X_v)
        acc = sum(y_pred == y_v)/len(y_v)
        acc_lst.append(acc)
    return acc_lst
```

In [271…

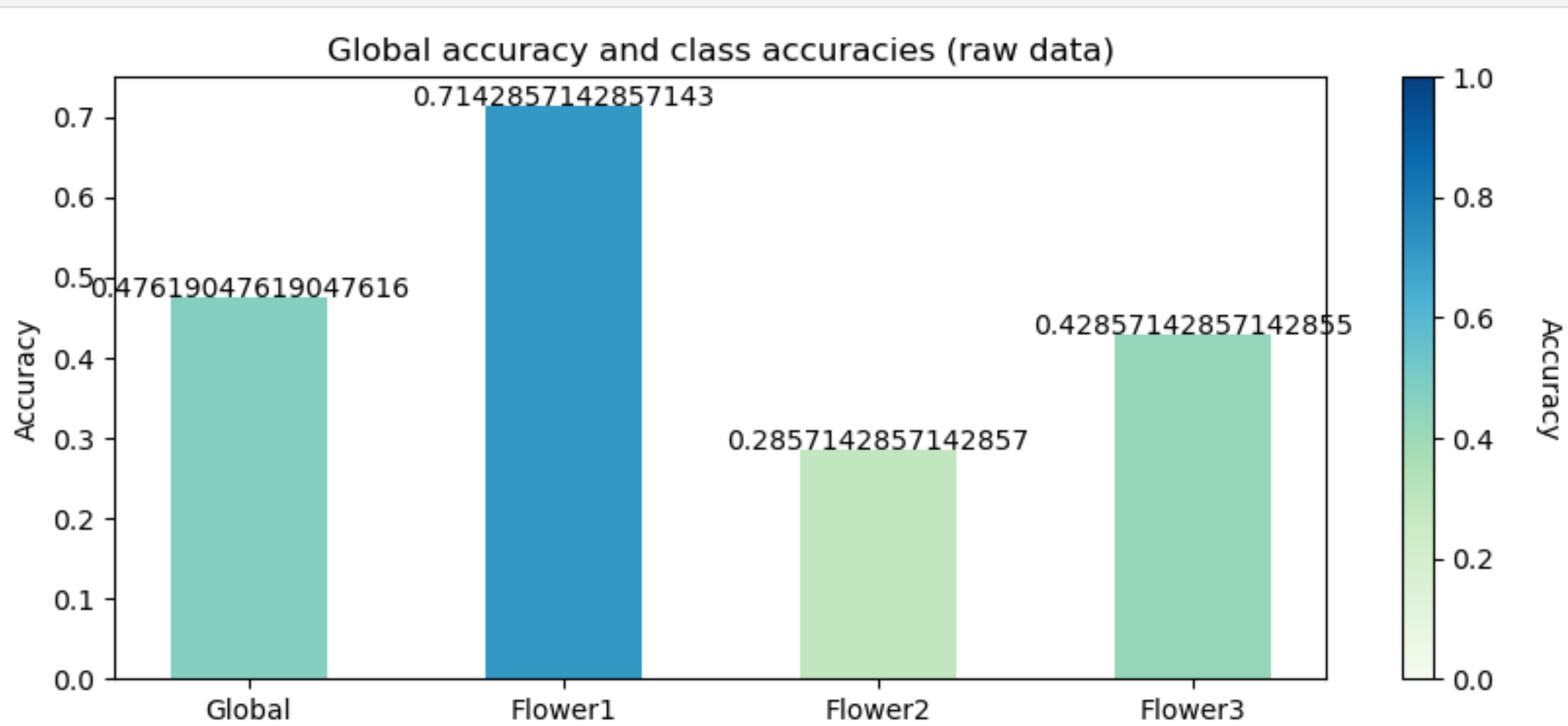
```
acc_or = test_diff_k_acc(X_train, y_train, X_val, y_val)
plt.plot(list(range(1,15)), acc_or,'-o')
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.title('Accuracy vs k (raw data)')
plt.show()
```



In [272…
```
neigh = KNeighborsClassifier(n_neighbors=14)
neigh.fit(X_train, y_train)
y_pred = neigh.predict(X_test)
```

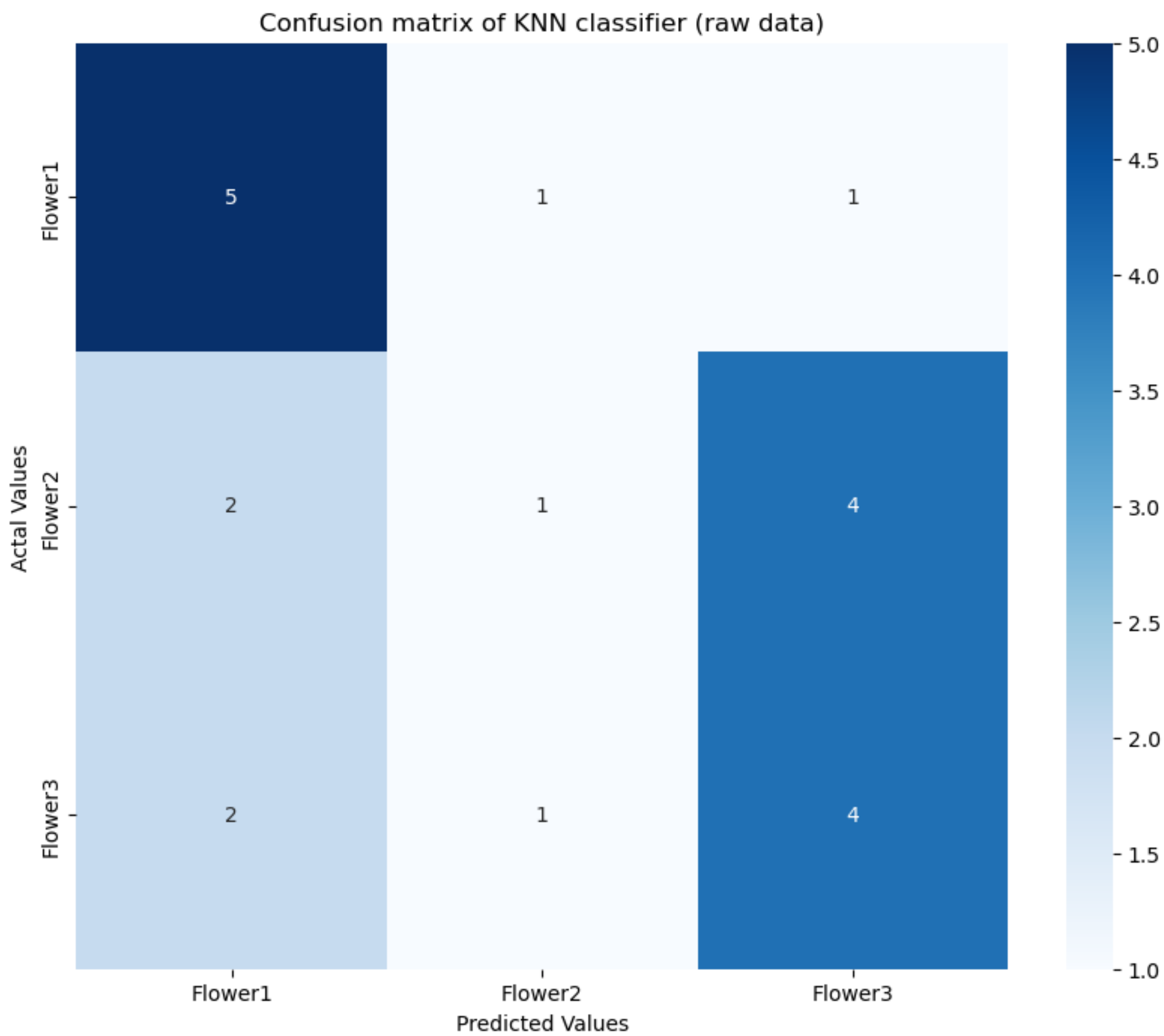
In [273…
```
def acc_lst(cm,c):
    global_acc = np.trace(cm)/np.sum(cm)
    class_acc_lst = []
    for i in range(c):
        class_acc = cm[i,i]/np.sum(cm,axis=1)[i]
        class_acc_lst.append(class_acc)
    accuracy = [global_acc, *class_acc_lst]
    return accuracy

def addlabels(x,y):
    for i in range(len(x)):
        plt.text(i, y[i], y[i], ha = 'center')

def pretty_bar(index, accuracy, title):
    fig, ax = plt.subplots(figsize=(10, 4))
    my_cmap = plt.cm.get_cmap('GnBu')
    colors = my_cmap(accuracy)
    rects = ax.bar(index, accuracy, color=colors, width=0.5)
    addlabels(index, accuracy)
    sm = matplotlib.cm.ScalarMappable(cmap=my_cmap)
    sm.set_array([])
    cbar = plt.colorbar(sm)
    cbar.set_label('Accuracy', rotation=270,labelpad=25)
    plt.title(title)
    plt.xticks(index)
    plt.ylabel('Accuracy')
    return plt.show()
```

In [274…
```
cm_raw = confusion_matrix(y_test, y_pred)
raw_acc = acc_lst(cm_raw, 3)
idxs = ['Global', 'Flower1', 'Flower2', 'Flower3']
pretty_bar(idxs, raw_acc, 'Global accuracy and class accuracies (raw data)')
```



In [230…

```
class_idx = ['Flower1', 'Flower2', 'Flower3']
cm_raw_df = pd.DataFrame(cm_raw,
                     index = class_idx,
                     columns = class_idx)
plt.figure(figsize=(10,8))
sns.heatmap(cm_raw_df, annot=True, cmap="Blues")
plt.title('Confusion matrix of KNN classifier (raw data)')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```



In [231…

```
precision_recall_fscore_support(y_test, y_pred)
```

Out[231]:

```
(array([0.55555556, 0.33333333, 0.44444444]),
 array([0.71428571, 0.14285714, 0.57142857]),
 array([0.625, 0.2  , 0.5  ]),
 array([7, 7, 7]))
```

In [136…

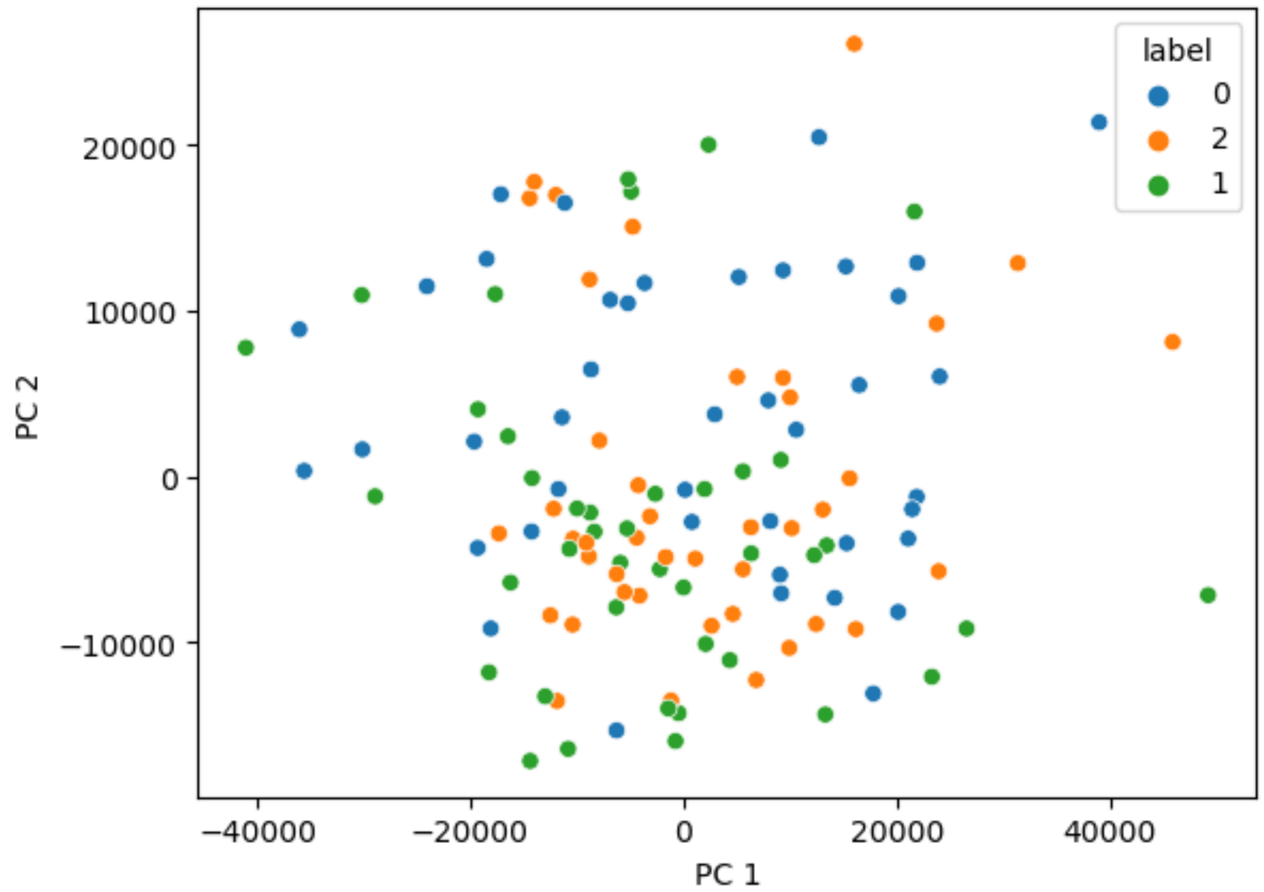
```
df = pd.concat([train_df, val_df, test_df], ignore_index=True)
df_f = df.drop(columns=['label'])
df_l = df['label'].copy()
```

In [137…

```
pca = PCA(n_components=2)
pca_2 = pca.fit_transform(df_f)
PC2_df = pd.DataFrame(data = pca_2,
                            columns = ['PC 1', 'PC 2'],
                            index = list(df_f.index)
                            )
PC2_df = pd.concat([PC2_df, df_l], axis = 1)
sns.scatterplot(data=PC2_df, x='PC 1',y='PC 2', hue="label")
```

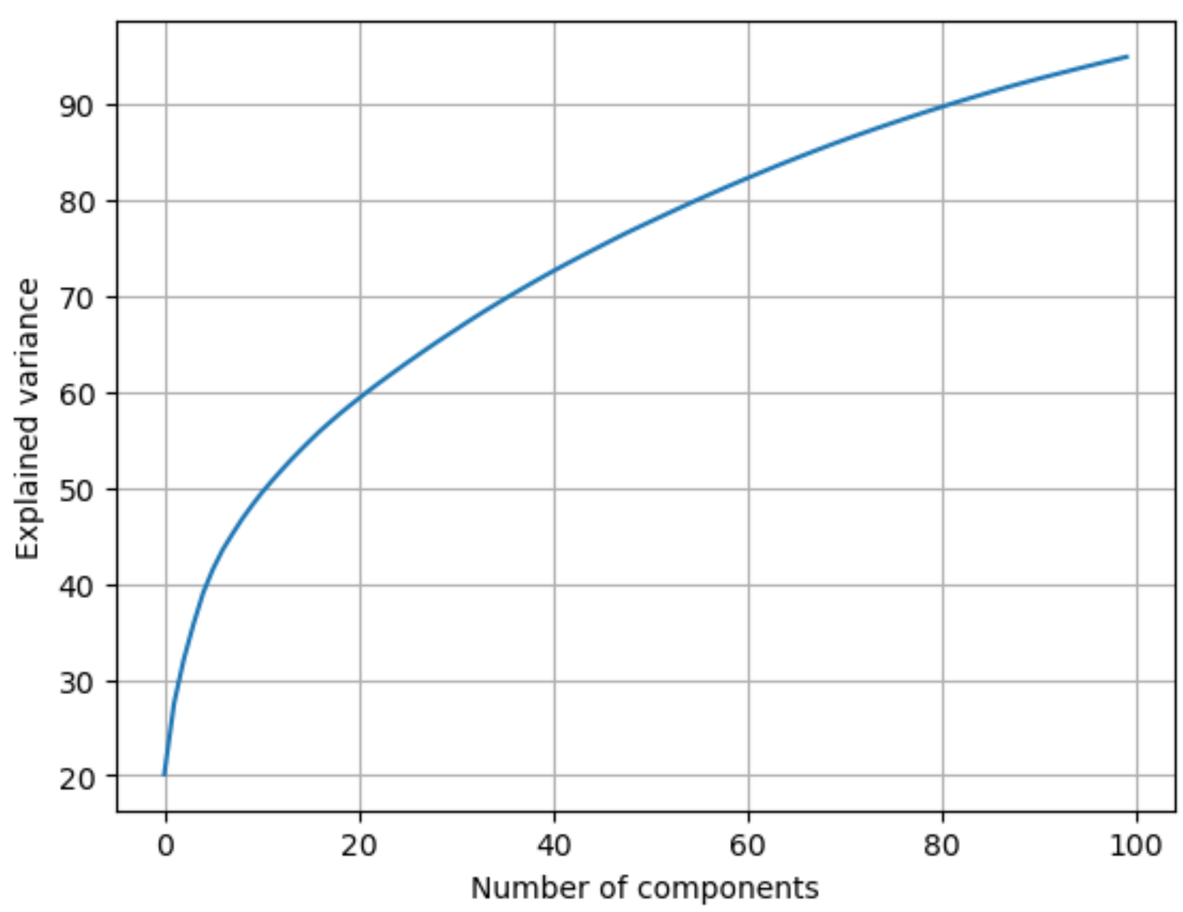
Out[137]:

```
<AxesSubplot:xlabel='PC 1', ylabel='PC 2'>
```



In [158…

```
pca_100 = PCA(n_components=100)
pca_100.fit(df_f)
plt.grid()
plt.plot(np.cumsum(pca_100.explained_variance_ratio_ * 100))
plt.xlabel('Number of components')
plt.ylabel('Explained variance')
plt.savefig('Scree plot.png')
```



In [284…

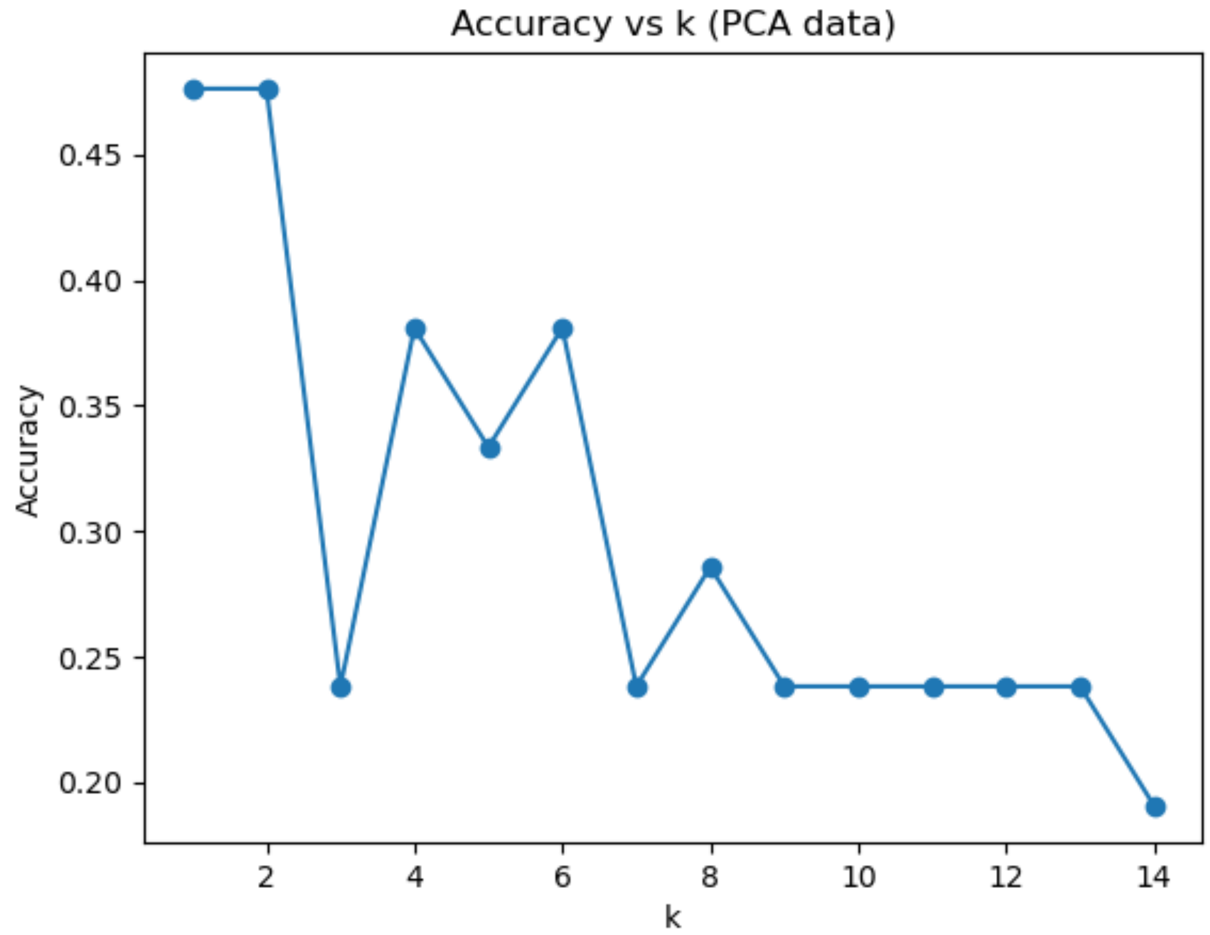
```
pca = PCA(n_components=80)
PC_80 = pca.fit_transform(df_f)
col_names = ['PC '+str(x) for x in range(1, 81)]
PC80_df = pd.DataFrame(data = PC_80,
                            columns = col_names,
                            index = list(df_f.index)
                            )
PC80_df = pd.concat([PC80_df, df_l], axis = 1)
```

In [285…

```
pca_train, pca_val, pca_test = PC80_df.iloc[:len(train_df)],  PC80_df.iloc[len(train_df):len(train_df)+len(val_df)],  PC80_df.iloc[len(train_df)+
X_train_pca, y_train, X_val_pca, y_val, X_test_pca, y_test = feature_label_split(pca_train, pca_val, pca_test)
```

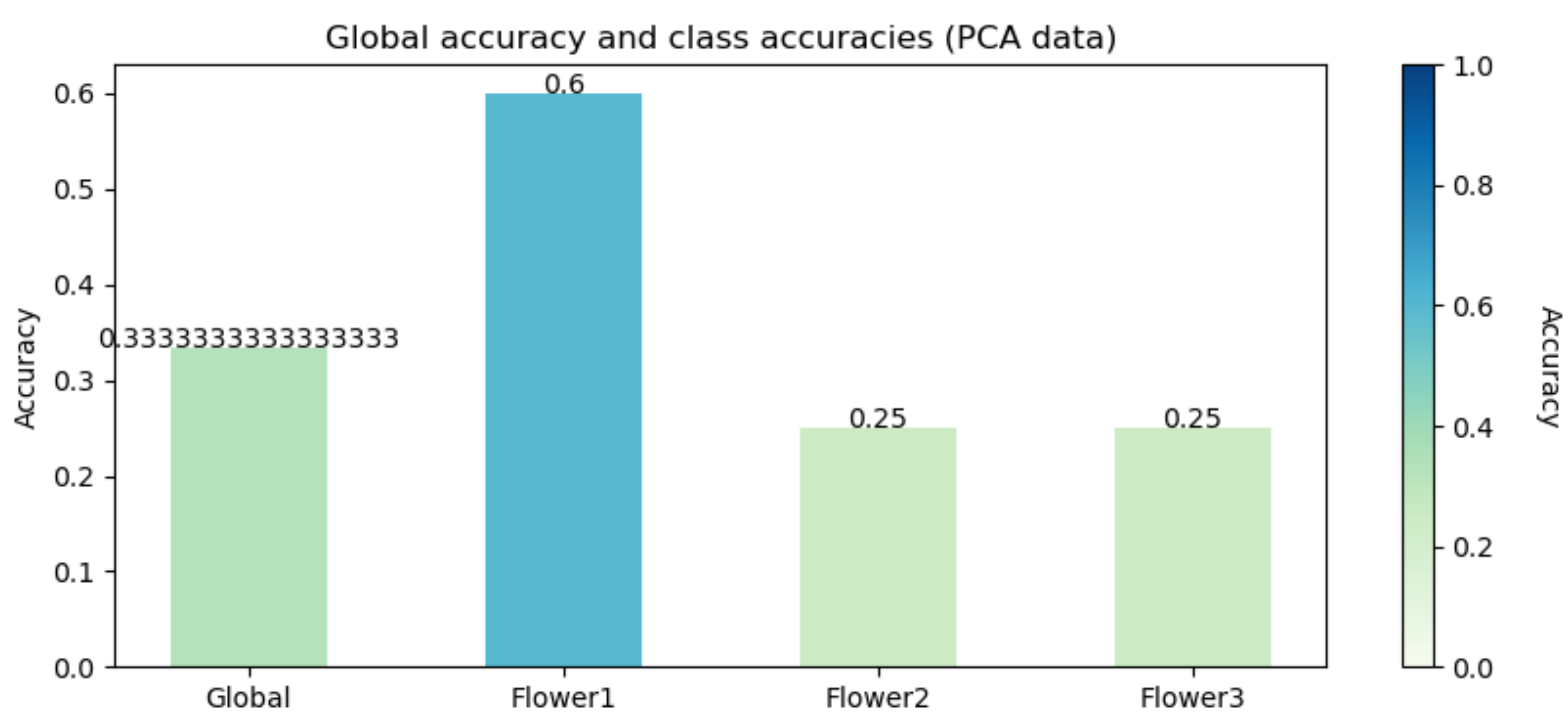
In [286…

```
acc_pca = test_diff_k_acc(X_train_pca, y_train, X_val_pca, y_val)
plt.plot(list(range(1,15)), acc_pca,'-o')
plt.xlabel("k")
plt.ylabel("Accuracy")
plt.title('Accuracy vs k (PCA data)')
plt.show()
```

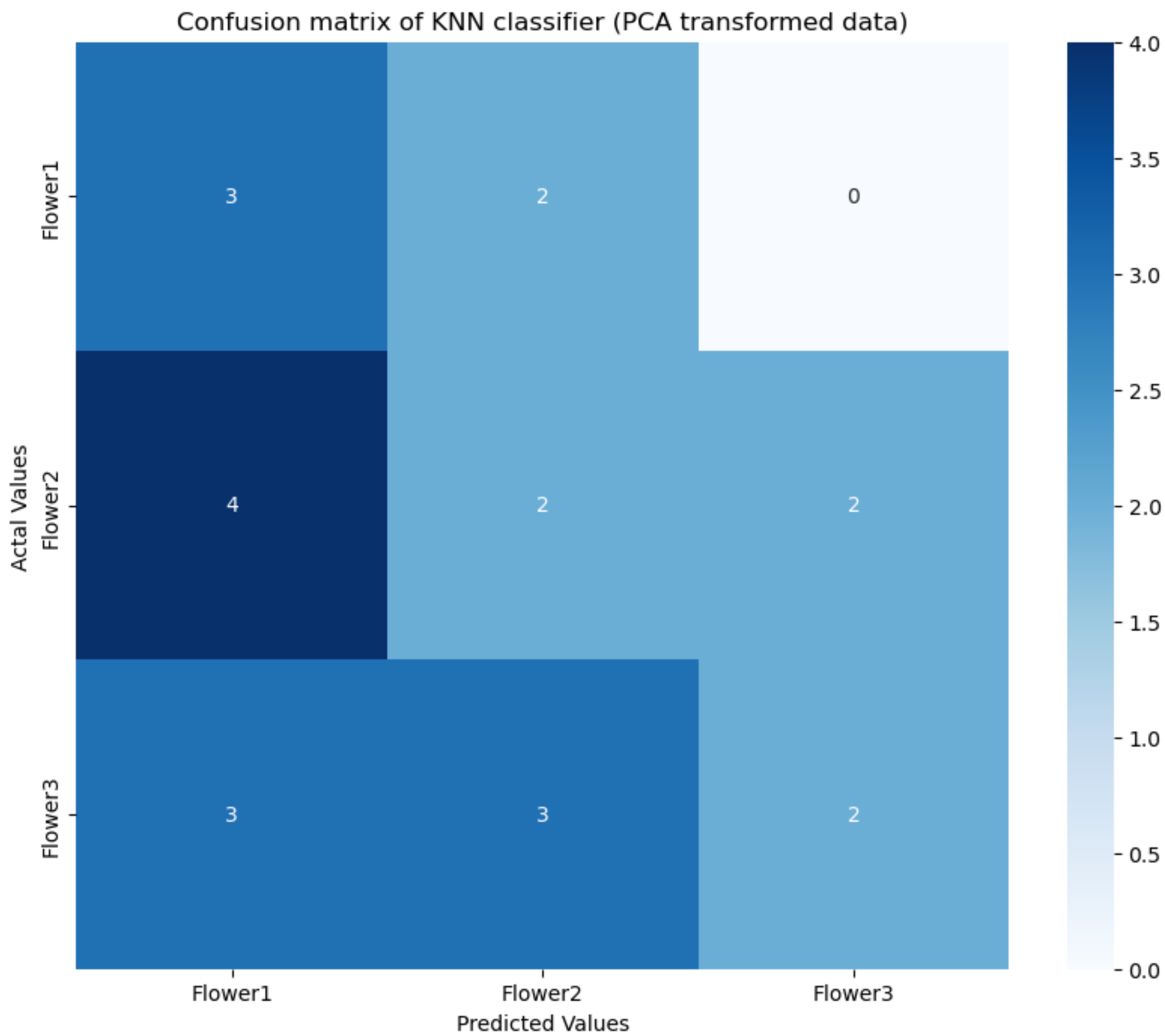



```
In [287…
neigh = KNeighborsClassifier(n_neighbors=2)
neigh.fit(X_train_pca, y_train)
y_pred_pca = neigh.predict(X_test_pca)
```

```
In [288…
cm_pca = confusion_matrix(y_test, y_pred_pca)
pca_acc = acc_lst(cm_pca, 3)
pretty_bar(idxs, pca_acc, 'Global accuracy and class accuracies (PCA data)')
```




```
In [290…
cm_pca_df = pd.DataFrame(cm_pca,
                     index = class_idx,
                     columns = class_idx)
plt.figure(figsize=(10,8))
sns.heatmap(cm_pca_df, annot=True, cmap="Blues")
plt.title('Confusion matrix of KNN classifier (PCA transformed data)')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```

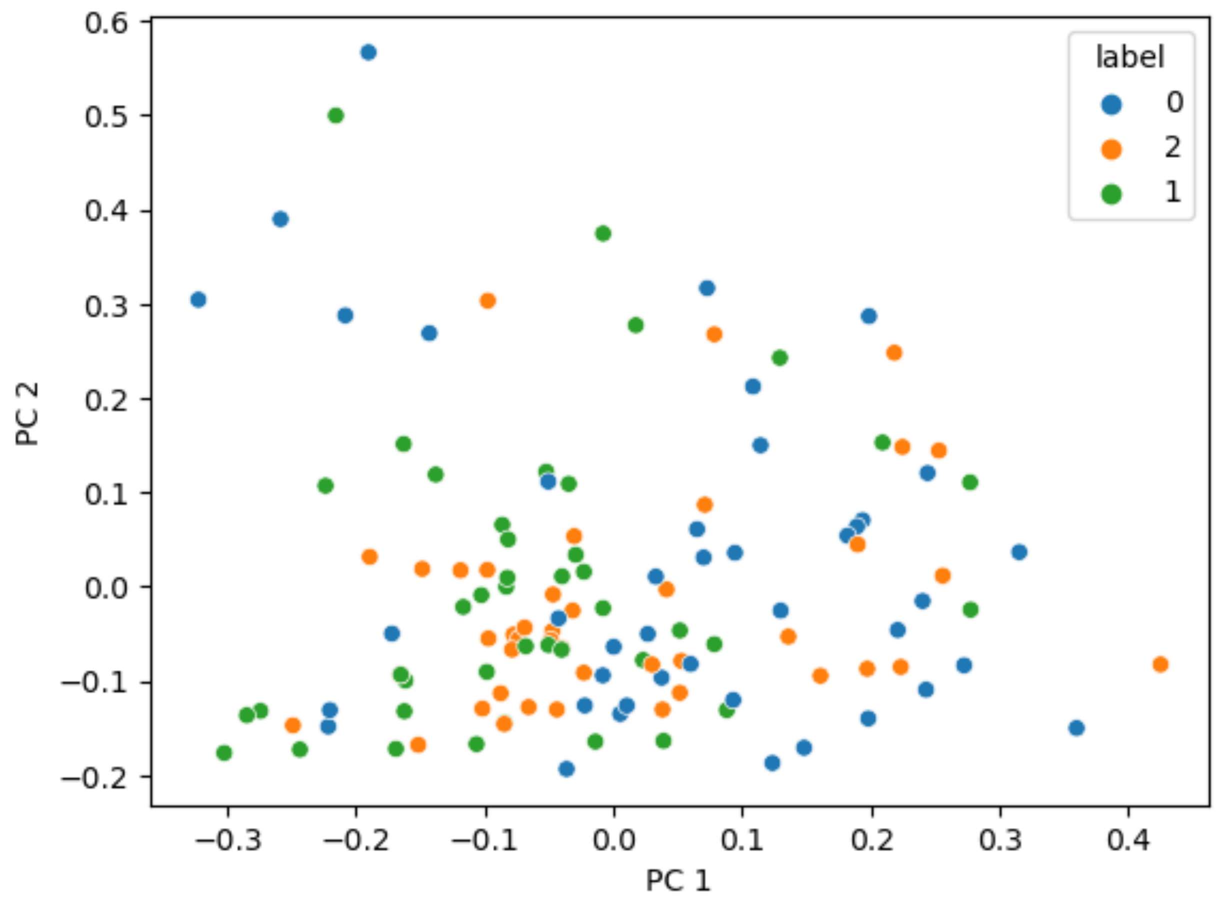


In [248…
```
precision_recall_fscore_support(y_test, y_pred_pca)
```

Out[248]:
```
(array([0.2       , 0.33333333, 0.5       ]),
 array([0.4  , 0.125, 0.5  ]),
 array([0.26666667, 0.18181818, 0.5       ]),
 array([5, 8, 8]))
```

In [252…
```
kpca2 = KernelPCA(kernel="cosine",
                  fit_inverse_transform=True,
                  gamma=10,
                  n_components=2)
kpca2 = kpca2.fit_transform(df_f)
col_names = ['PC '+str(x) for x in range(1, 3)]
kpa2_df = pd.DataFrame(data = kpca2,
                               columns = col_names,
                               index = list(df_f.index)
                               )
kpa2_df = pd.concat([kpa2_df, df_l], axis = 1)
sns.scatterplot(data=kpa2_df, x='PC 1',y='PC 2', hue="label")
```

Out[252]:
```
<AxesSubplot:xlabel='PC 1', ylabel='PC 2'>
```

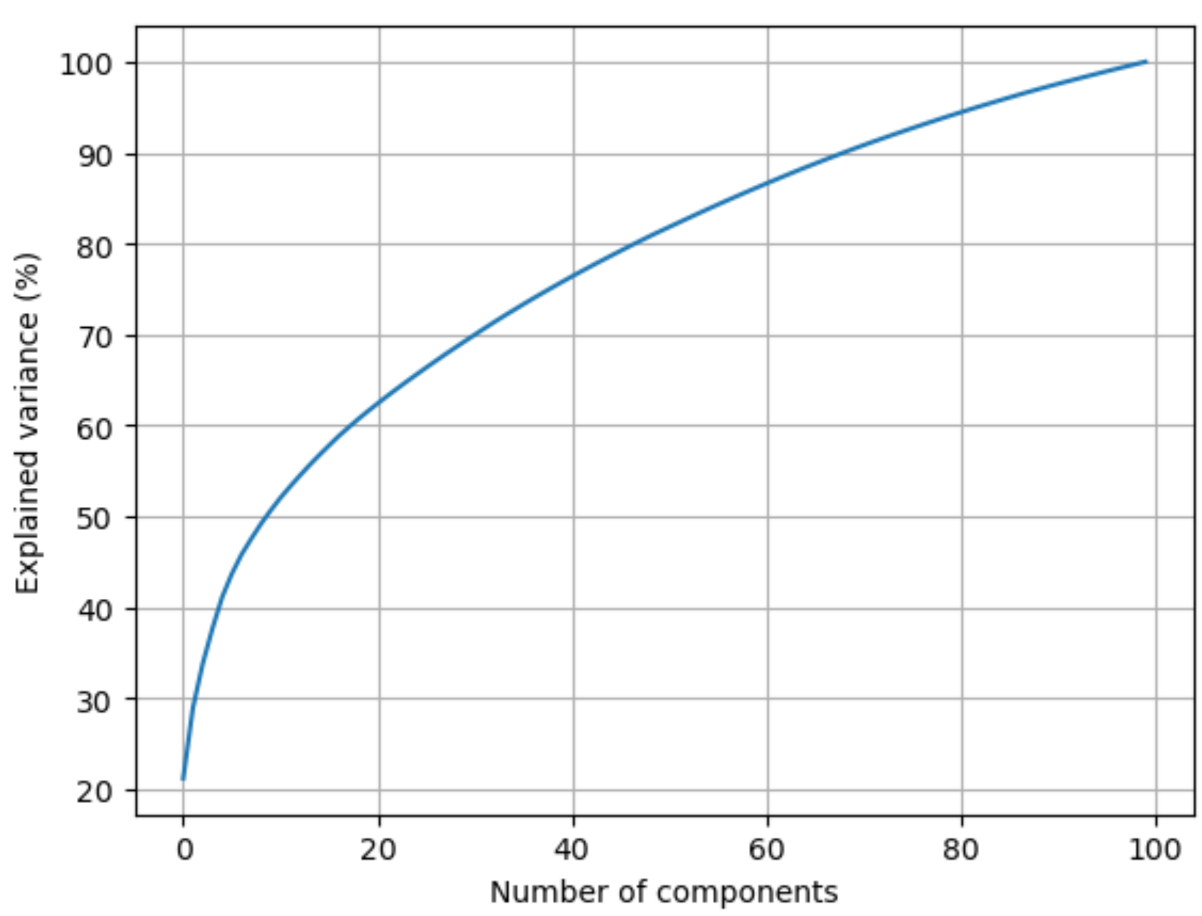


In [250…

```
kpca_100 = KernelPCA(n_components=100)
kpca_100.fit(df_f)
plt.grid()
variance_ratio = np.cumsum(kpca_100.lambdas_) / np.sum(kpca_100.lambdas_)
plt.plot(variance_ratio * 100)
plt.xlabel('Number of components')
plt.ylabel('Explained variance (%)')
```

Out[250]:

```
Text(0, 0.5, 'Explained variance (%)')
```



In [278…

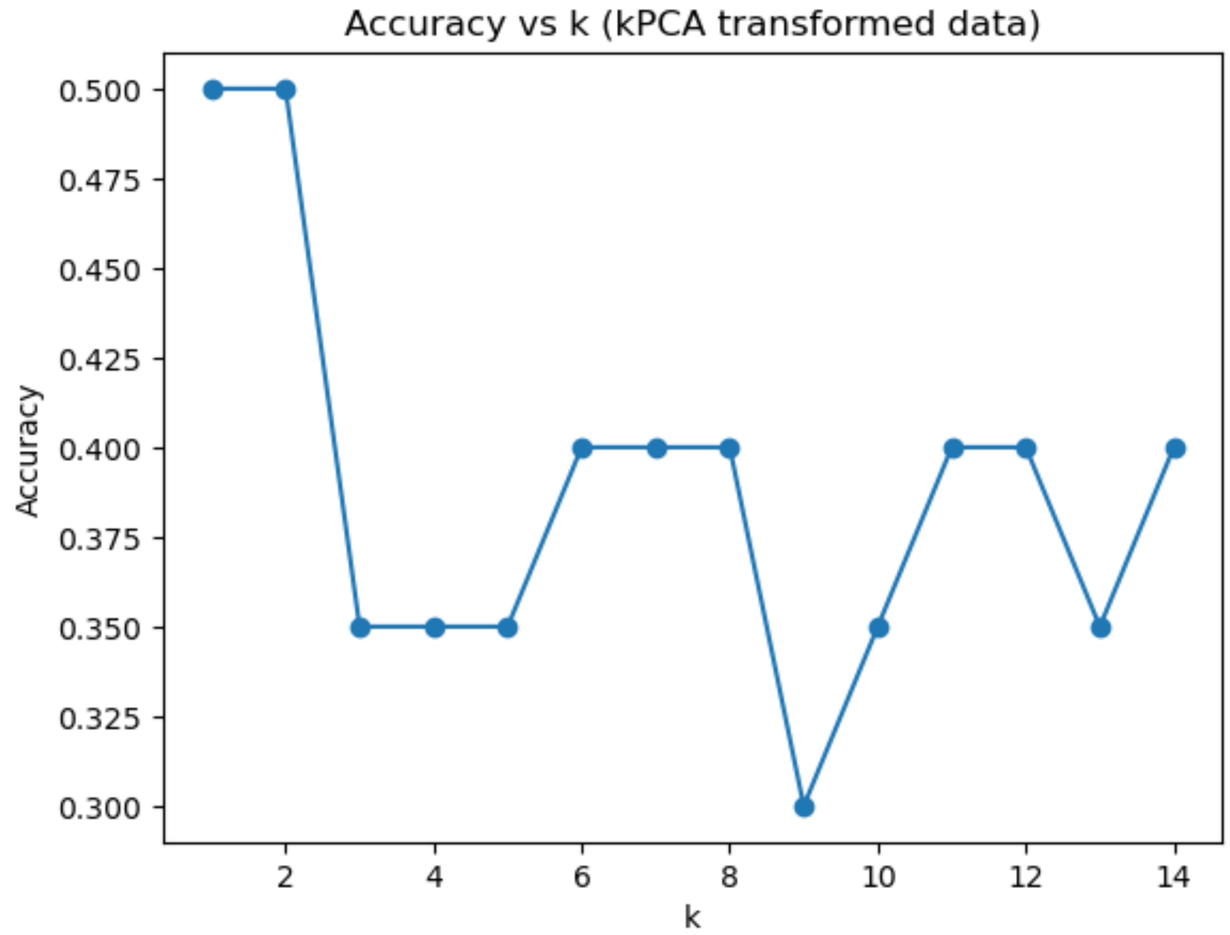
```
kpca68 = KernelPCA(kernel="cosine",
                 fit_inverse_transform=True,
                 gamma=10,
                 n_components=68)
kpca68 = kpca68.fit_transform(df_f)
col_names = ['PC '+str(x) for x in range(1, 69)]
kpca68_df = pd.DataFrame(data = kpca68,
                               columns = col_names,
                               index = list(df_f.index)
                               )
kpca68_df = pd.concat([kpca68_df, df_l], axis = 1)
```

In [279…

```
kpca_train, kpca_val, kpca_test = kpca68_df.iloc[:80],  kpca68_df.iloc[80:100],  kpca68_df.iloc[100:]
X_train_kpca, y_train, X_val_kpca, y_val, X_test_kpca, y_test = feature_label_split(kpca_train, kpca_val, kpca_test)
```

In [280…

```
acc_kpca = test_diff_k_acc(X_train_kpca, y_train, X_val_kpca, y_val)
plt.plot(list(range(1,15)), acc_kpca,'-o')
plt.xlabel("k")
plt.ylabel("Accuracy")
plt.title('Accuracy vs k (kPCA transformed data)')
plt.show()
```

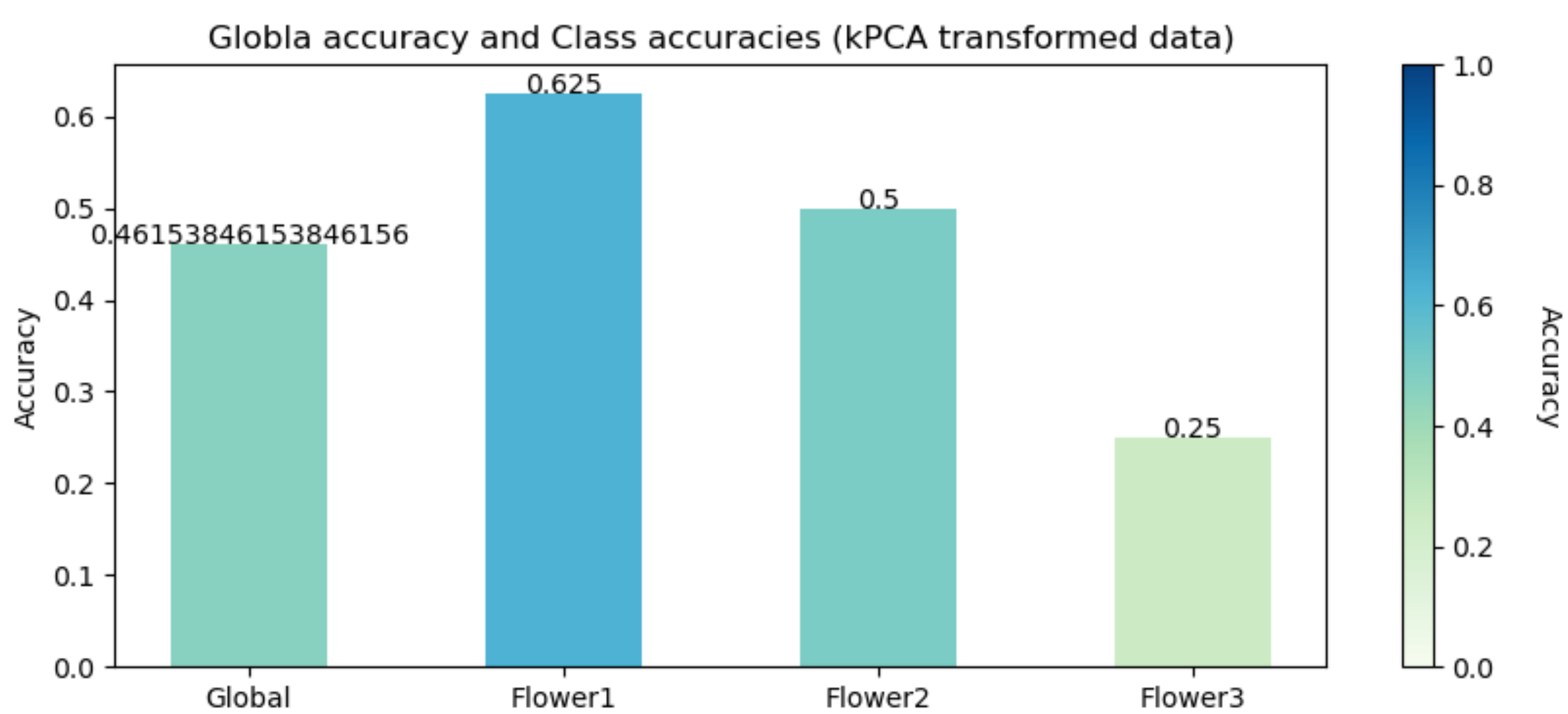


In [281…

```
neigh = KNeighborsClassifier(n_neighbors=2)
neigh.fit(X_train_kpca, y_train)
y_pred_kpca = neigh.predict(X_test_kpca)
```

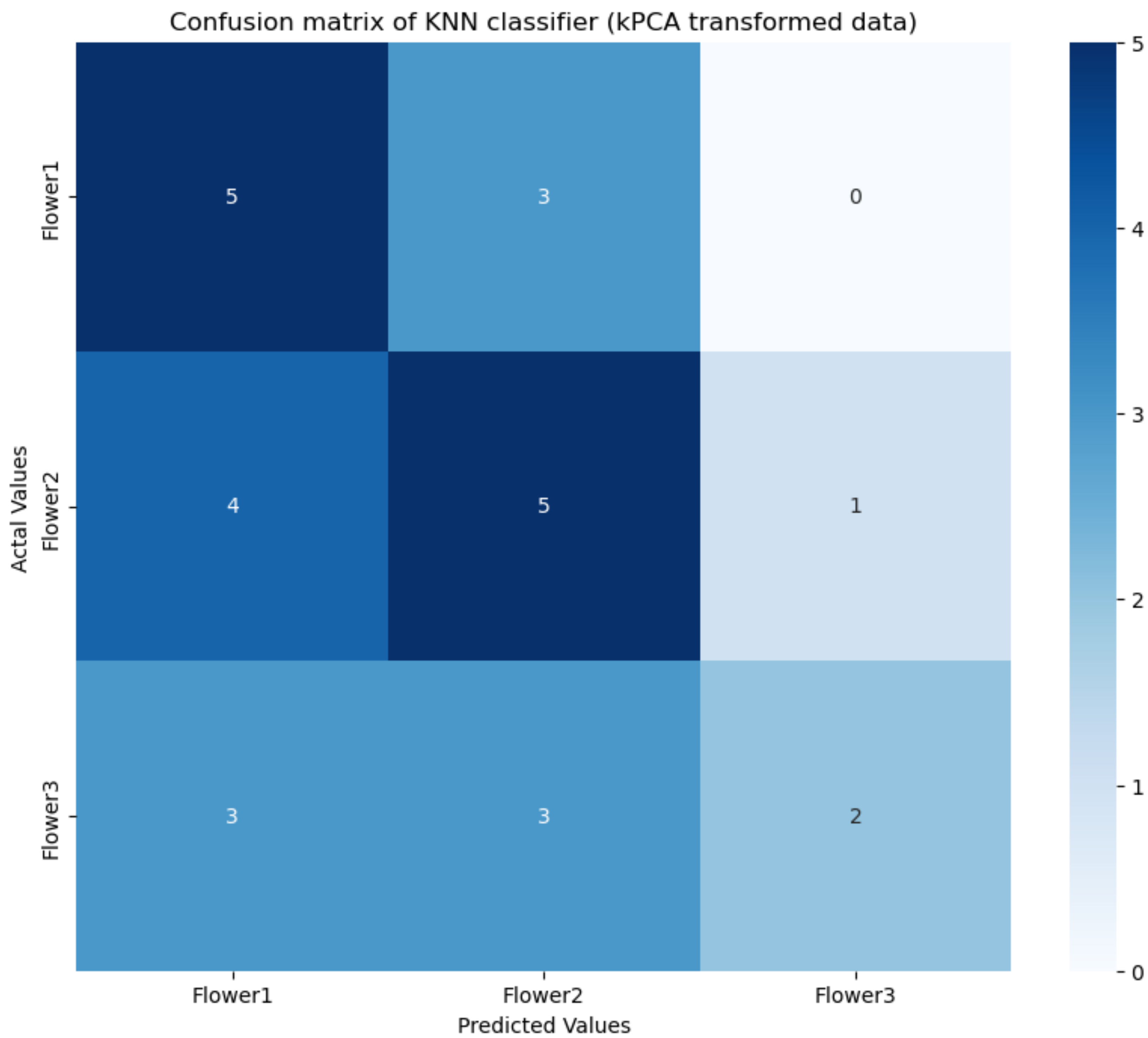
In [282…

```
cm_kpca = confusion_matrix(y_test, y_pred_kpca)
kpca_acc = acc_lst(cm_kpca, 3)
pretty_bar(idxs, kpca_acc,'Globla accuracy and Class accuracies (kPCA transformed data)')
```



In [283…

```
cm_kpca_df = pd.DataFrame(cm_kpca,
                     index = class_idx,
                     columns = class_idx)
plt.figure(figsize=(10,8))
sns.heatmap(cm_kpca_df, annot=True, cmap="Blues")
plt.title('Confusion matrix of KNN classifier (kPCA transformed data)')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```

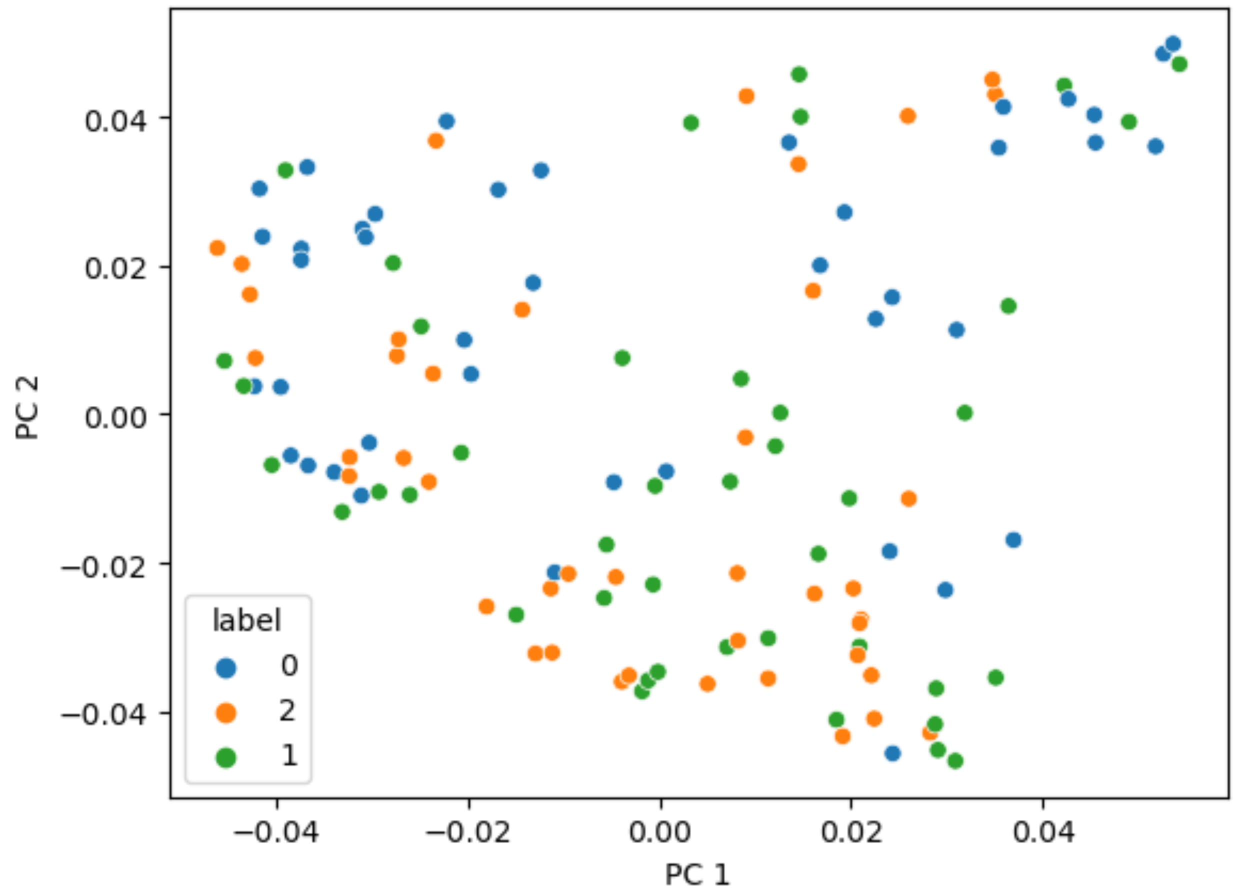


In [265…
```
precision_recall_fscore_support(y_test, y_pred_kpca)
```
Out[265]:
```
(array([0.4       , 0.66666667, 0.375     ]),
 array([0.75 , 0.2  , 0.375]),
 array([0.52173913, 0.30769231, 0.375     ]),
 array([ 8, 10,  8]))
```

In [266…
```
df_arr = df_f.to_numpy()
mle = skdim.id.MLE().fit(df_arr)
print(mle.dimension_)
```
```
15.991258110042095
```

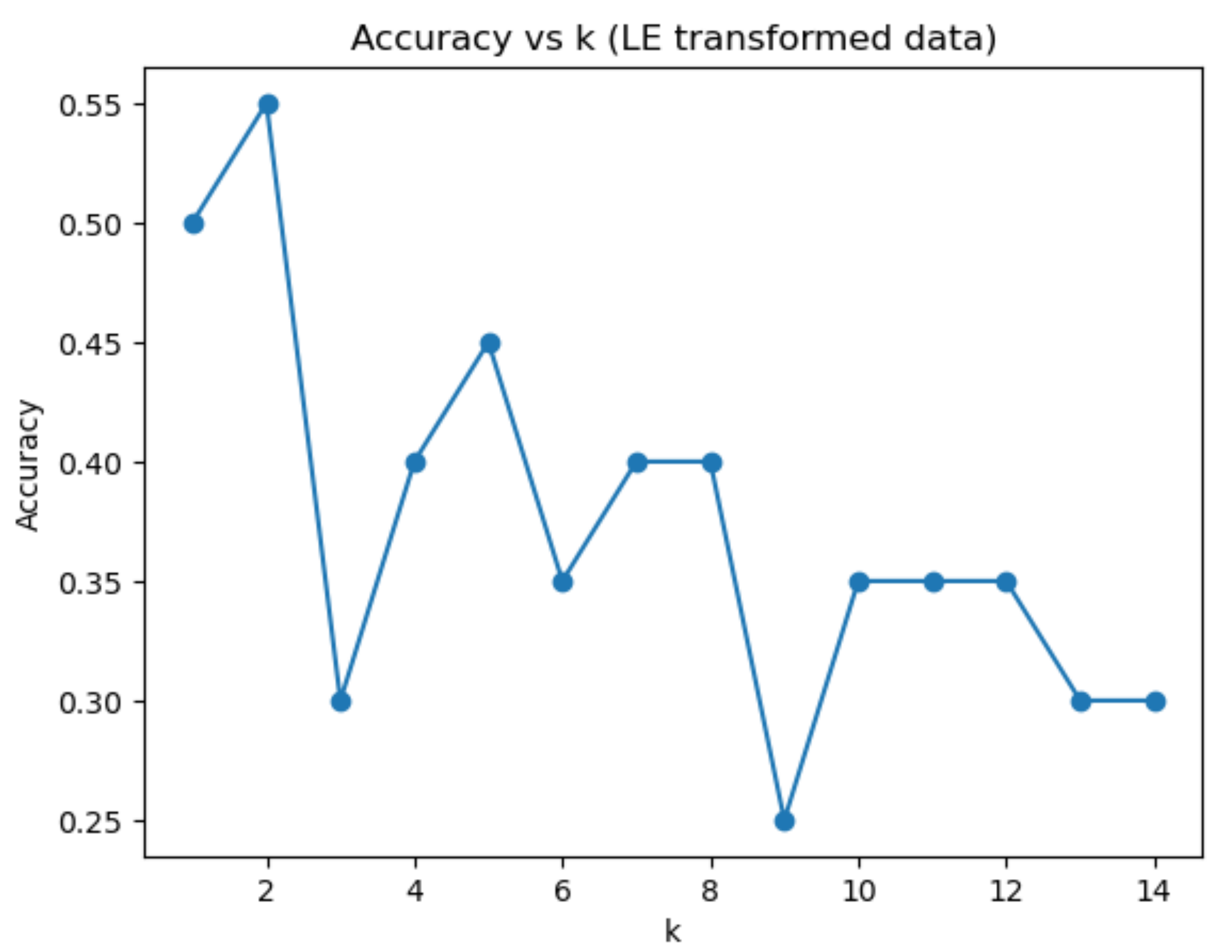
In [297…
```
LE = SpectralEmbedding(n_components=2)
LE2 = LE.fit_transform(df_f)
LE2_df = pd.DataFrame(data = LE2,
                             columns = ['PC 1', 'PC 2'],
                             index = list(df_f.index)
                             )
LE2_df = pd.concat([LE2_df, df_l], axis = 1)
```

In [298…
```
sns.scatterplot(data=LE2_df,x='PC 1',y='PC 2', hue="label")
```
Out[298]:
```
<AxesSubplot:xlabel='PC 1', ylabel='PC 2'>
```



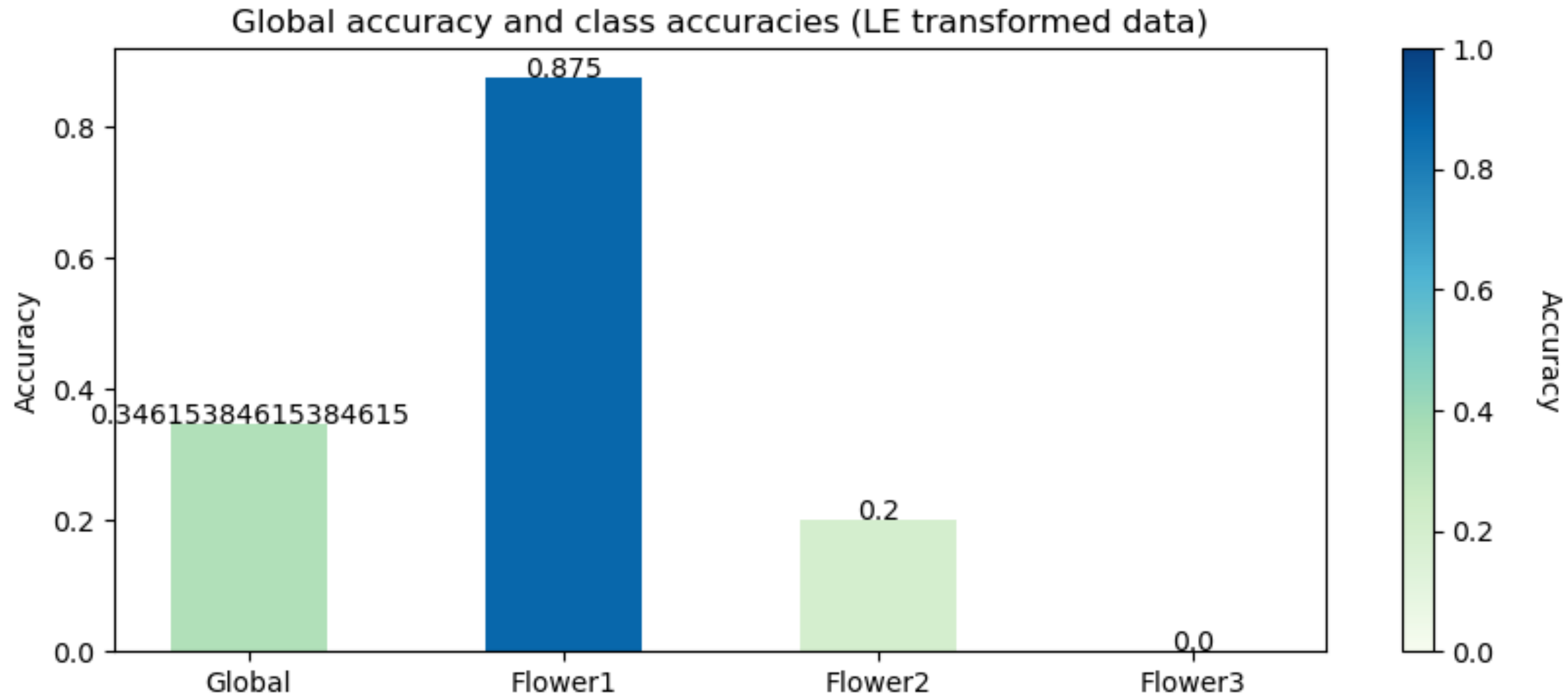
In [299…
```
LE_train, LE_val, LE_test = LE2_df.iloc[:80],  LE2_df.iloc[80:100],  LE2_df.iloc[100:]
X_train_LE, y_train, X_val_LE, y_val, X_test_LE, y_test = feature_label_split(LE_train, LE_val, LE_test)
```

In [300…
```
acc_LE = test_diff_k_acc(X_train_LE, y_train, X_val_LE, y_val)
plt.plot(list(range(1,15)), acc_LE,'-o')
plt.xlabel("k")
plt.ylabel("Accuracy")
plt.title('Accuracy vs k (LE transformed data)')
plt.show()
```

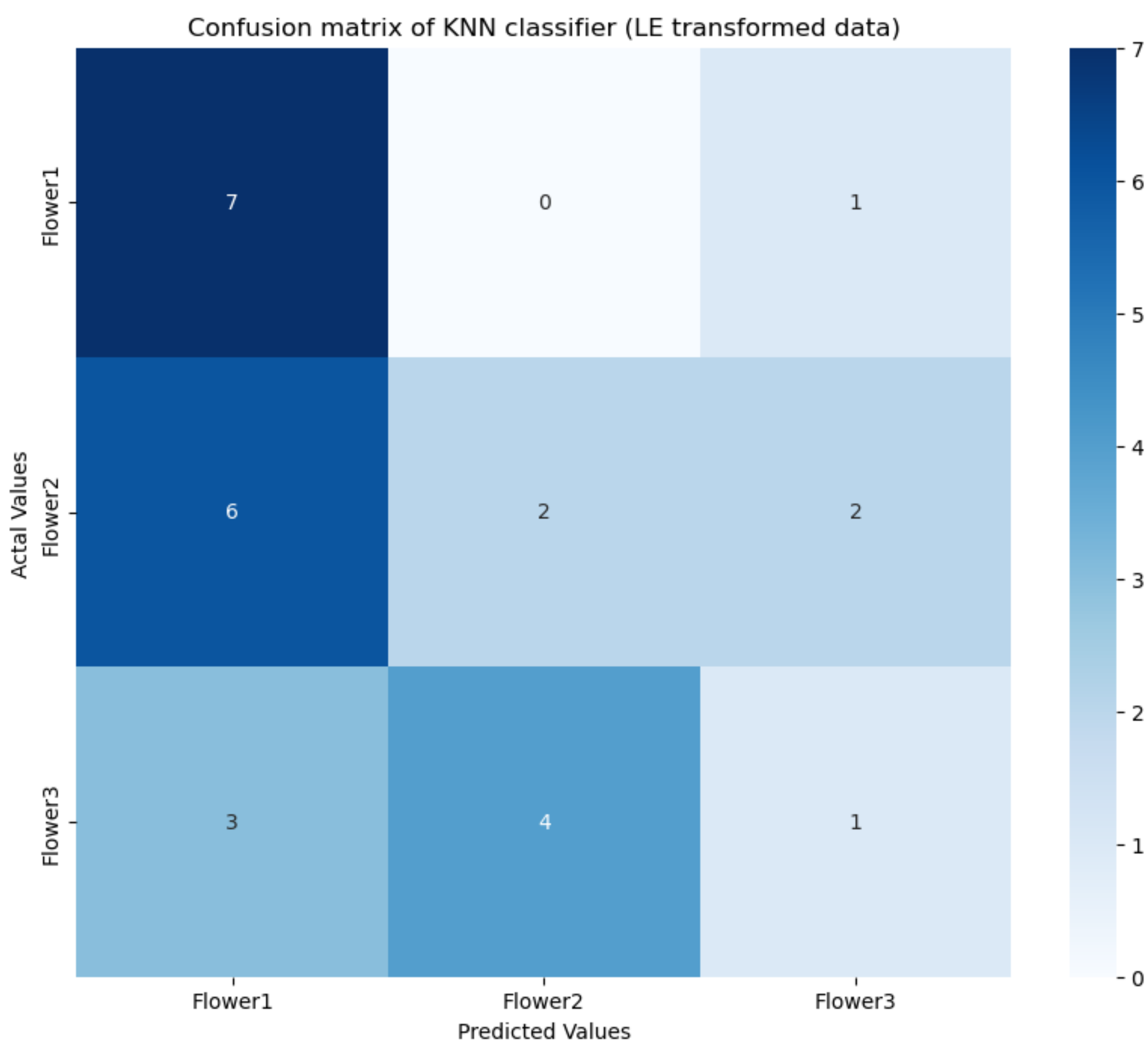


In [309…
```
neigh = KNeighborsClassifier(n_neighbors=5)
neigh.fit(X_train_LE, y_train)
y_pred_LE = neigh.predict(X_test_LE)
```

In [310…
```
cm_le = confusion_matrix(y_test, y_pred_LE)
le_acc = acc_lst(cm_le, 3)
pretty_bar(idxs, le_acc,'Global accuracy and class accuracies (LE transformed data)')
```



In [305…
```
cm_le_df = pd.DataFrame(cm_le,
                        index = class_idx,
                        columns = class_idx)
plt.figure(figsize=(10,8))
sns.heatmap(cm_le_df, annot=True, cmap="Blues")
plt.title('Confusion matrix of KNN classifier (LE transformed data)')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```

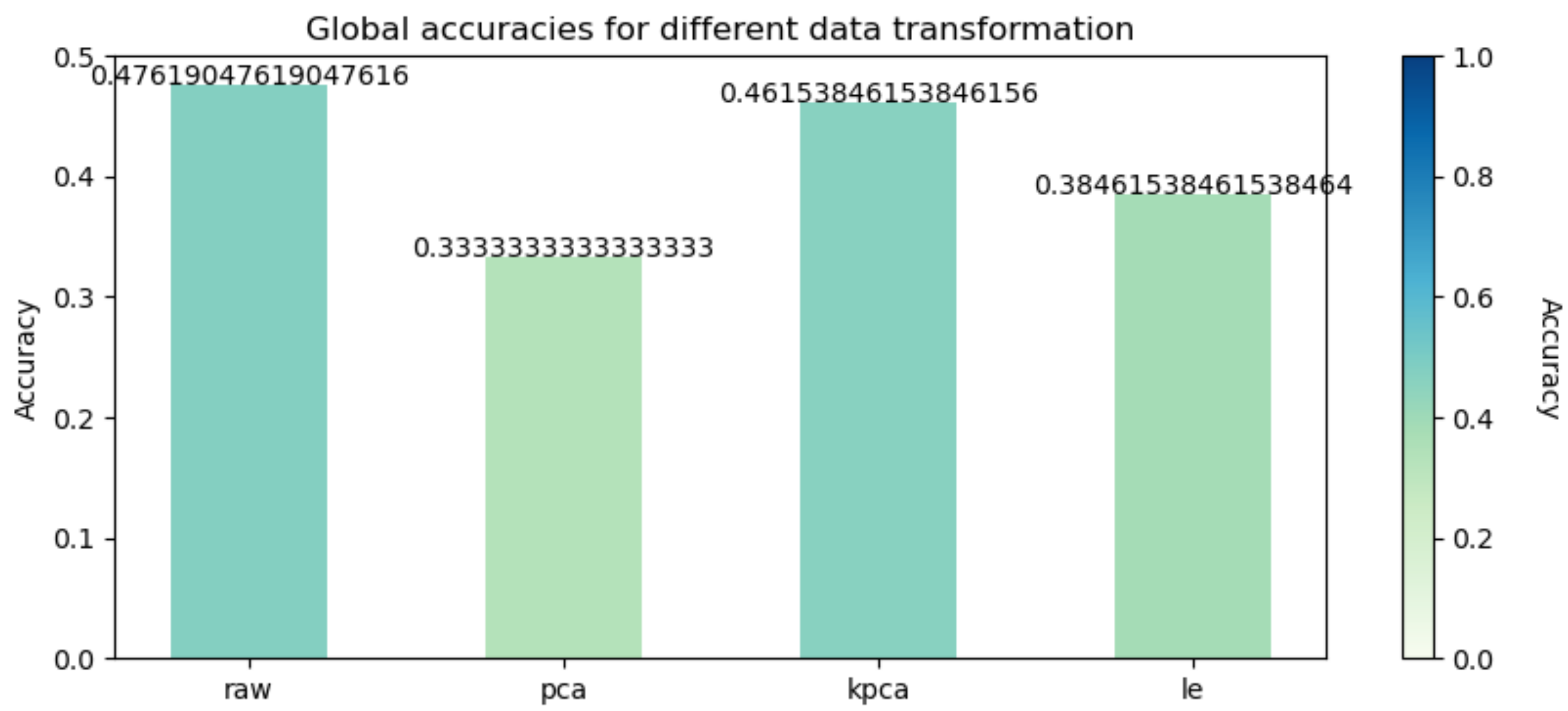


In [306…
```
precision_recall_fscore_support(y_test, y_pred_LE)
```

Out[306]:
```
(array([0.4375    , 0.33333333, 0.25      ]),
 array([0.875, 0.2  , 0.125]),
 array([0.58333333, 0.25      , 0.16666667]),
 array([ 8, 10,  8]))
```

In [307…
```
cms = [cm_raw, cm_pca, cm_kpca, cm_le]
global_acc_lst = []
for cm in cms:
    global_acc = np.trace(cm)/np.sum(cm)
    global_acc_lst.append(global_acc)
```

In [308…
```
dmtech_lst = ['Raw', 'PCA', 'kPCA', 'LE']
pretty_bar(dmtech_lst, global_acc_lst, 'Global accuracies for different data transformation')
```



In [ ]: