In [18]:
```
# Import packages I may need.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy as sp
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
```

In [36]:
```
# A) Load the data, report sizes of training and testing sets. How many class 0,
# class 1, and class 2 samples are in the training set and testing set, respectively?
ftrain = 'Data/pca_train.csv'
ftest = 'Data/pca_test.csv'
traindata = pd.read_csv(ftrain, header=0)
testdata = pd.read_csv(ftest, header=0)
trainDF = pd.DataFrame(traindata)
testDF = pd.DataFrame(testdata)

print('The training dataset has a size of ' + str(trainDF.size) + ', and the',
      'testing dataset has a size of '+ str(testDF.size) +'. \n')

# Get class counts
# Train set: 37 Class 0, 51 Class 1, 37 Class 2
trcount1 = (trainDF.Class == 0).sum()
trcount2 = (trainDF.Class == 1).sum()
trcount3 = (trainDF.Class == 2).sum()

# Test set: 22 Class 0, 20 Class 1, 11 Class 2
testcount1 = (testDF.Class == 0).sum()
testcount2 = (testDF.Class == 1).sum()
testcount3 = (testDF.Class == 2).sum()

print('The training set consists of the following number of class entries: \n'
      'Class 0: ' + str(trcount1) + '\n'
      'Class 1: ' + str(trcount2) + '\n'
      'Class 2: ' + str(trcount3) + '\n')
print('The testing set consists of the following numbers of class entries: \n'
      'Class 0: ' + str(testcount1) + '\n'
      'Class 1: ' + str(testcount2) + '\n'
      'Class 2: ' + str(testcount3) + '\n')
```

```
The training dataset has a size of 1750, and the testing dataset has a size of 742.

The training set consists of the following number of class entries:
Class 0: 37
Class 1: 51
Class 2: 37

The testing set consists of the following numbers of class entries:
Class 0: 22
Class 1: 20
Class 2: 11
```

In [20]:
```
# B) Run normalization on all input features in both datasets. (use min/max of each
# column in the training set to normalize the testing set, and do NOT normalize
# the output "Class".

def normalize(df):
    for column in df.columns:
        if column != 'Class':
            hi = df[column].max()
            lo = df[column].min()
            df[column] = (df[column] - lo) / (hi - lo)
    return df


norm_trainDF = normalize(trainDF)
norm_testDF = normalize(testDF)
traintarget = norm_trainDF['Class']
testtarget = pd.DataFrame(norm_testDF['Class'])
testtarget.rename(columns={'Class': 'Target Class'}, inplace=True)
```

In [21]:
```
# B.i) Calculate the covariance matrix of the NEW training dataset. Specify 1) the
# dimension of the resulted covariance matrix and 2) please report the first 5 rows
# and 5 columns of the entire covariance matrix.
norm_trainDF.drop(columns='Class', inplace=True)
norm_testDF.drop(columns='Class', inplace=True)
traincov = norm_trainDF.cov()
print('The covariance matrix has ' + str(traincov.shape[0]) + ' rows and '
      + str(traincov.shape[1])
      + ' columns, making it ' + str(traincov.shape[0]) + '-dimensional.\n')

sample = traincov.iloc[0:5, 0:5]
print(sample.to_string())
print('\n')
```

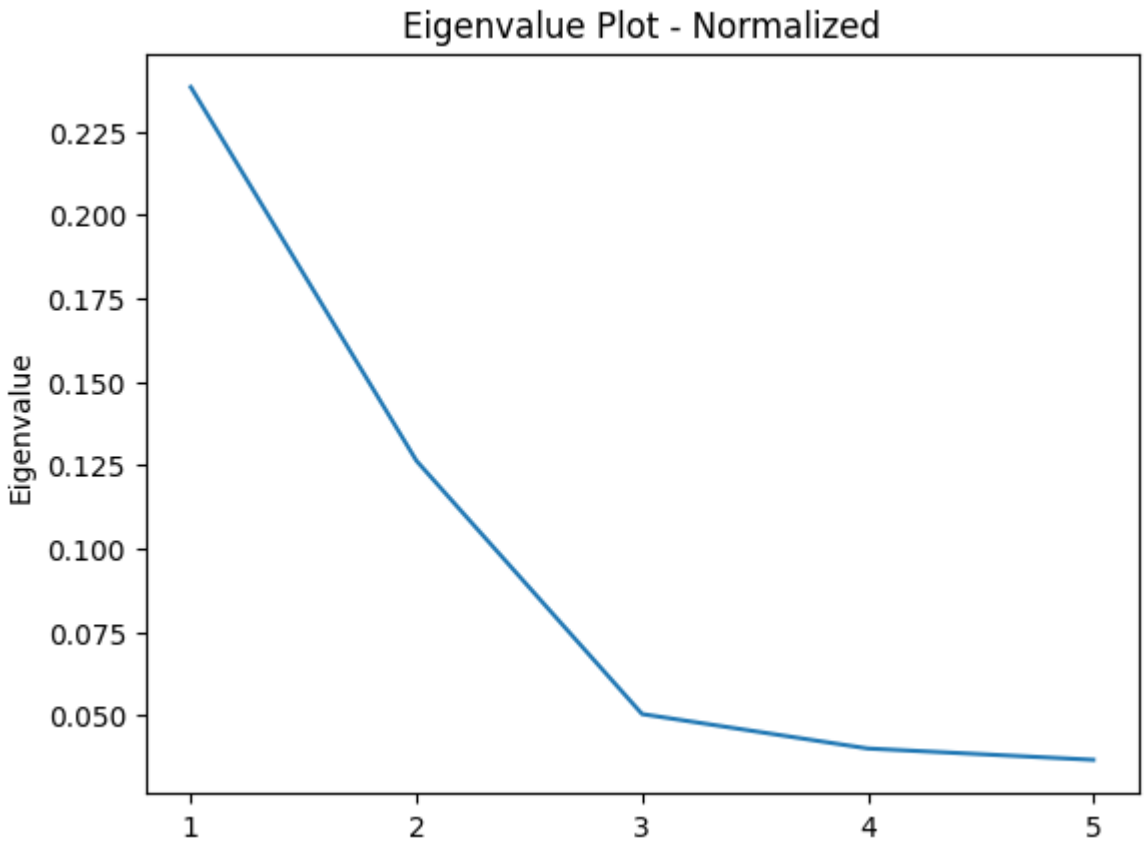
```
The covariance matrix has 13 rows and 13 columns, making it 13-dimensional.

                   alcohol  malic_acid       ash  alcalinity_of_ash  magnesium
alcohol           0.054379    0.005361  0.005785          -0.014173   0.016116
malic_acid        0.005361    0.052094  0.006839           0.011951  -0.000757
ash               0.005785    0.006839  0.022047           0.011179   0.011012
alcalinity_of_ash -0.014173    0.011951  0.011179           0.029989  -0.003111
magnesium          0.016116   -0.000757  0.011012          -0.003111   0.038562
```

In [22]:
```
# B.ii) Calculate the eigenvalues and eigenvectors based on the entire covariance
# matrix. Report size of covariance matrix and its 5 largest eigenvalues.
eigs, vects = np.linalg.eig(traincov.astype(float))
largeeigs = sorted(eigs, reverse=True)[0:5]
print('The normalized covariance matrix has size ' + str(traincov.size) + '.\n')
print('Five largest eigenvalues, normalized: ' + str(largeeigs) + '\n')
```

```
The normalized covariance matrix has size 169.

Five largest eigenvalues, normalized: [0.23832336476046118, 0.12628503466563168, 0.05
035845207718296, 0.04003988048566645, 0.03664120361873715]
```

In [39]:
```
# B.iii) Display the eigenvalues using a bar graph, and choose a reasonable number of
```

```
# eigenvectors. Justify answer.
eigplot = plt.figure()
num = ['1', '2', '3', '4', '5']
plt.plot(num, largeeigs)
plt.title('Eigenvalue Plot - Normalized')
plt.ylabel('Eigenvalue')
plt.show()

print('I would choose three eigenvectors, as the eigenvalues appear to nearly\n' +
      'flatten out after the first three. In other words, the eigenvectors\n' +
      'corresponding to the three largest eigenvalues contain the bulk of\n' +
      'the information useful for our PCA analysis.')
```




```
I would choose three eigenvectors, as the eigenvalues appear to nearly
flatten out after the first three. In other words, the eigenvectors
corresponding to the three largest eigenvalues contain the bulk of
the information useful for our PCA analysis.
```

In [24]:

```
# B.iv) Combine PCA with a K-nearest neighbor (KNN) classifier. PCA will reduce
# dimensionality of original data into p principal components and KNN (K=5,
# use Euclidean distance) will  be employed to the p principal components for
# classification.
# B.iv.a) Report accuracy with p=5, generate .csv file with results
pca = PCA(n_components=5)
trainresult = pca.fit_transform(norm_trainDF)
trainresultDF = pd.DataFrame(trainresult)
testresult = pca.transform(norm_testDF)
testresultDF = pd.DataFrame(testresult, columns=['PC1', 'PC2', 'PC3', 'PC4', 'PC5'])

knn = KNN(n_neighbors=5, metric='euclidean')
knn.fit(trainresultDF, traintarget)
prediction = knn.predict(testresultDF)
```

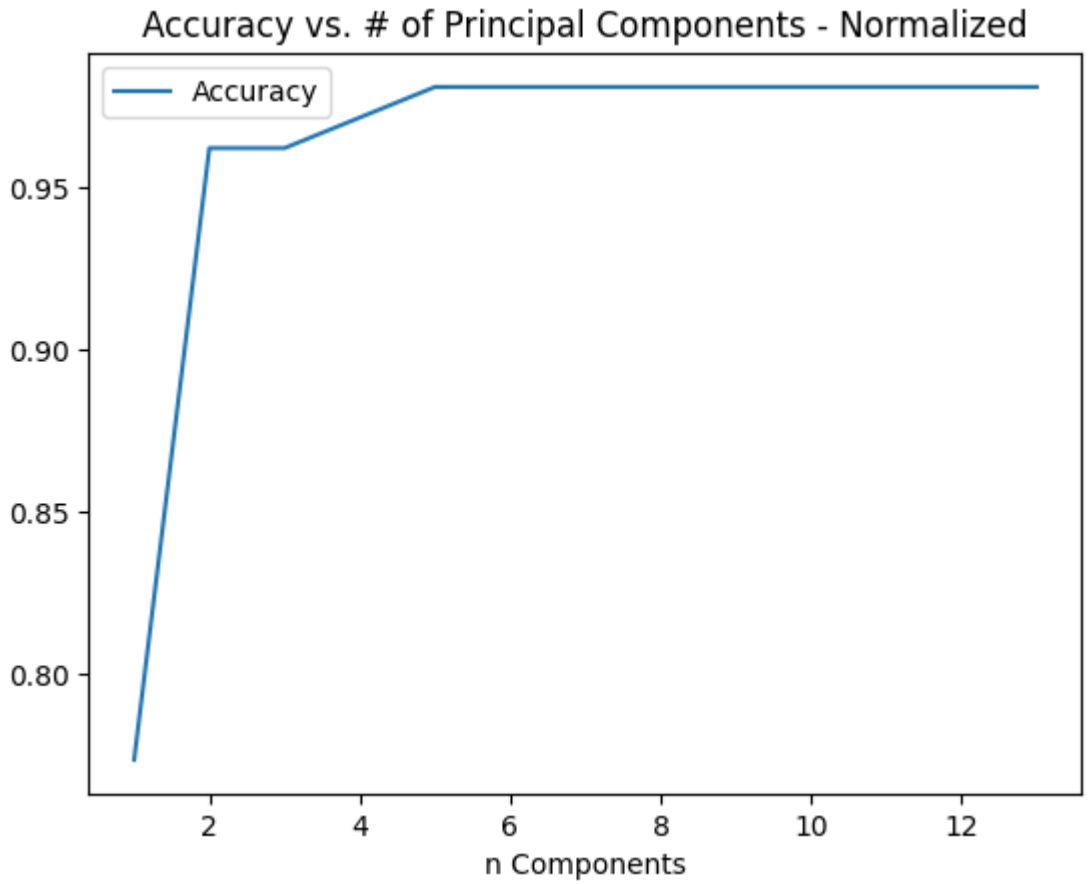
```
prediction = pd.DataFrame(prediction, columns=['Predicted Class'])
results = pd.concat([testresultDF, testtarget, prediction], axis=1)
acc = accuracy_score(testtarget, prediction)
print('The accuracy using the standardized data with 5 principal components and 5'
      + ' nearest \n' 'neighbors is ' + str(acc) + '.' +
      'The file demonstrating this output is titled \n' +
      '\'Normalized PCA Results.csv\'.')
results.to_csv('Normalized PCA Results.csv', index=False)
```

```
The accuracy using the standardized data with 5 principal components and 5 nearest
neighbors is 0.9811320754716981.The file demonstrating this output is titled
'Normalized PCA Results.csv'.
```

```
C:\Users\epurn\PycharmProjects\CSC522\venv\lib\site-packages\sklearn\base.py:443: Use
rWarning: X has feature names, but KNeighborsClassifier was fitted without feature na
mes
  warnings.warn(
```

In [25]:

```
# B.iv.b) Plot results with varying number of principal components.
def plot_results(trainset, testset, traintarg, testtarg):
    plist = [1, 2, 3, 5, 10, 13]
    results = []
    for p in plist:
        entry = []
        entry.append(p)

        pca = PCA(n_components=p)
        trainresult = pca.fit_transform(trainset)
        trainresultDF = pd.DataFrame(trainresult)
        testresult = pca.transform(testset)
        testresultDF = pd.DataFrame(testresult)

        knn = KNN(n_neighbors=5, metric='euclidean')
        knn.fit(trainresultDF, traintarg)
        prediction = knn.predict(testresultDF)
        prediction = pd.DataFrame(prediction)
        acc = accuracy_score(testtarg, prediction)

        entry.append(acc)
        results.append(entry)
    results = pd.DataFrame(results, columns=['n Components', 'Accuracy'])
    return results

points = plot_results(norm_trainDF, norm_testDF, traintarget, testtarget)

points.plot(x='n Components', y='Accuracy')
plt.title('Accuracy vs. # of Principal Components - Normalized')
plt.show()
```



In [40]:
```
# B.iv.c) What is the most reasonable number of principal components?
print('I would choose 5, as there does not seem to be any improvement in accuracy\n' +
      'beyond that point. These results suggest that the first 5 principal \n' +
      'components explain the vast majority of the variance observed in the \n' +
      'dataset.')
```

```
I would choose 5, as there does not seem to be any improvement in accuracy
beyond that point. These results suggest that the first 5 principal
components explain the vast majority of the variance observed in the
dataset.
```

In [27]:
```
# C) Repeat all of the above, except standardize the data instead of normalize it.
# Standardize with sklearn tools
sc = StandardScaler()
sc.fit(trainDF)
std_train = sc.transform(trainDF)
std_trainDF = pd.DataFrame(std_train, columns=norm_trainDF.columns)

sc.fit(testDF)
std_test = sc.transform(testDF)
std_testDF = pd.DataFrame(std_test, columns=norm_testDF.columns)
```

In [28]:
```
# C.i) Calculate the covariance matrix of the NEW training dataset. Specify 1) the
# dimension of the resulted covariance matrix and 2) please report the first 5 rows
# and 5 columns of the entire covariance matrix.
traincov2 = std_trainDF.cov()
print('The covariance matrix has ' + str(traincov2.shape[0]) + ' rows and '
      + str(traincov2.shape[1]) + ' columns, making it '
      + str(traincov2.shape[0]) + '-dimensional.\n')
```

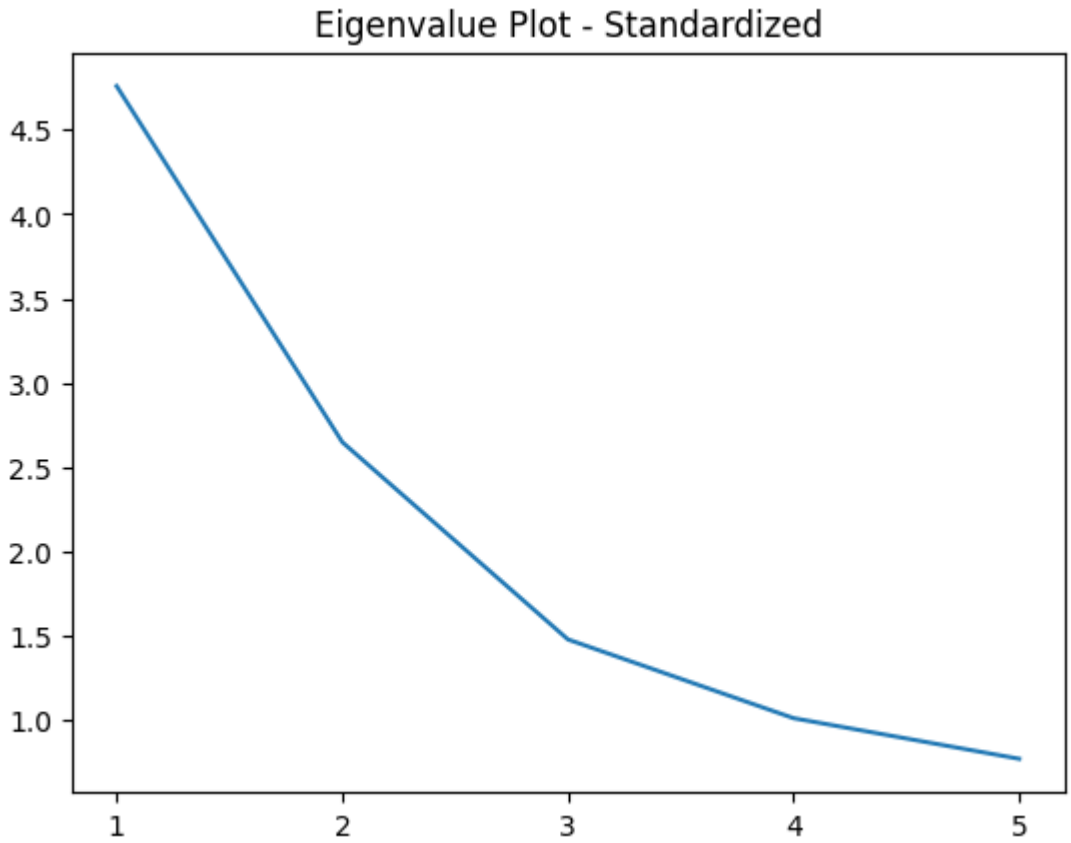
```
sample2 = traincov.iloc[0:5, 0:5]
print(sample2.to_string())
print('\n')
```

```
The covariance matrix has 13 rows and 13 columns, making it 13-dimensional.

                   alcohol  malic_acid       ash  alcalinity_of_ash  magnesium
alcohol           0.054379    0.005361  0.005785          -0.014173   0.016116
malic_acid        0.005361    0.052094  0.006839           0.011951  -0.000757
ash               0.005785    0.006839  0.022047           0.011179   0.011012
alcalinity_of_ash -0.014173    0.011951  0.011179           0.029989  -0.003111
magnesium         0.016116   -0.000757  0.011012          -0.003111   0.038562
```

In [29]:
```
# C.ii) Calculate the eigenvalues and eigenvectors based on the entire covariance
# matrix. Report size of covariance matrix and its 5 largest eigenvalues.
eigs2, vects2 = np.linalg.eig(traincov2.astype(float))
largeeigs2 = sorted(eigs2, reverse=True)[0:5]
print('The standardized covariance matrix has size ' + str(traincov2.size) + '.\n')
print('Five largest eigenvalues, standardized: ' + str(largeeigs2) + '\n')
```

```
The standardized covariance matrix has size 169.

Five largest eigenvalues, standardized: [4.759134655006307, 2.6488837008249857, 1.479
6208845395895, 1.0129897334928872, 0.7721873479912539]
```

In [42]:
```
# C.iii) Display the eigenvalues using a bar graph, and choose a reasonable number of
# eigenvectors. Justify answer.
eigplot2 = plt.figure()
num2 = ['1', '2', '3', '4', '5']
plt.plot(num2, largeeigs2)
plt.title('Eigenvalue Plot - Standardized')
plt.show()

print('I would choose at least five eigenvectors, as the eigenvalues do not \n' +
      'appear to flatten out on the curve shown above. This leads me to \n' +
      'believe that the first 5 principal components do not full explain\n' +
      'the variance.')
```



I would choose at least five eigenvectors, as the eigenvalues do not appear to flatten out on the curve shown above. This leads me to believe that the first 5 principal components do not full explain the variance.

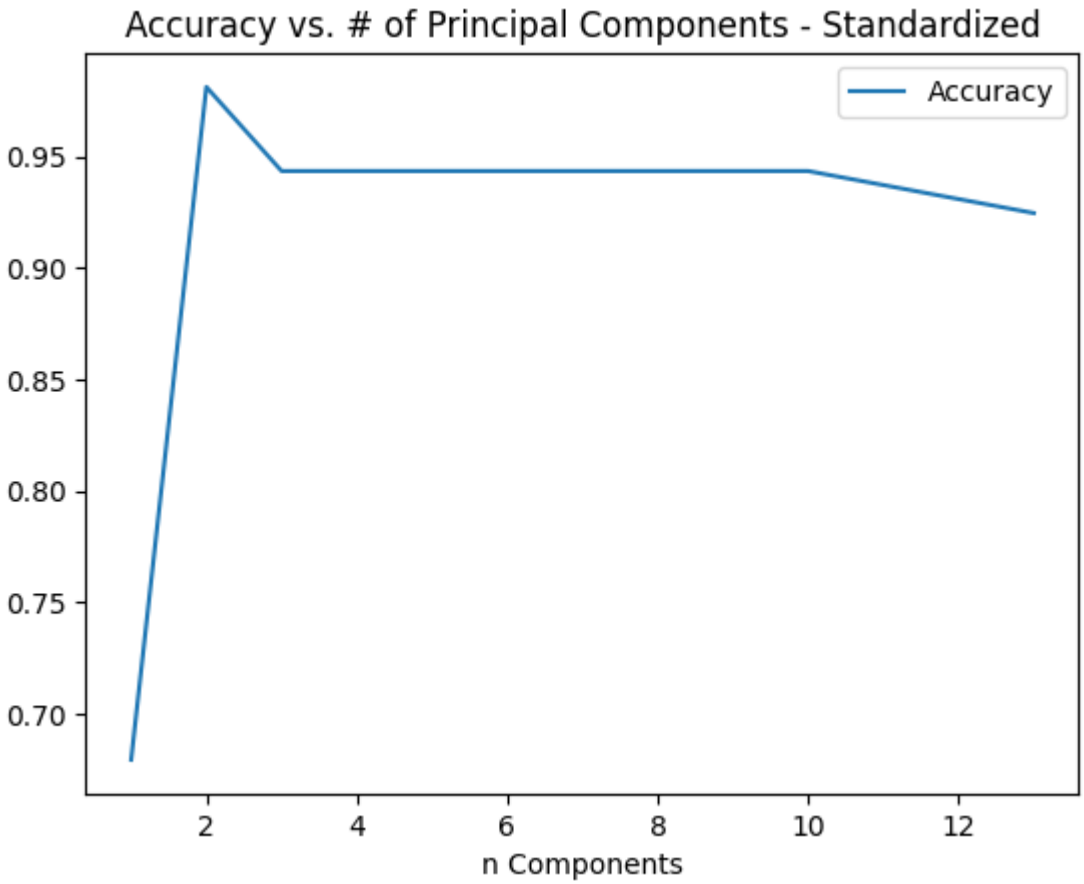
In [43]:
```
# C.iv) Combine PCA with a K-nearest neighbor (KNN) classifier. PCA will reduce
# dimensionality of original data into p principal components and KNN (K=5,
# use Euclidean distance) will  be employed to the p principal components for
# classification.
# C.iv.a
pca = PCA(n_components=5)
trainresult = pca.fit_transform(std_trainDF)
trainresultDF = pd.DataFrame(trainresult)
testresult = pca.transform(std_testDF)
testresultDF = pd.DataFrame(testresult, columns=['PC1', 'PC2', 'PC3', 'PC4', 'PC5'])

knn = KNN(n_neighbors=5, metric='euclidean')
knn.fit(trainresultDF, traintarget)
prediction = knn.predict(testresultDF)
prediction = pd.DataFrame(prediction, columns=['Predicted Class'])
results = pd.concat([testresultDF, testtarget, prediction], axis=1)
acc2 = accuracy_score(testtarget, prediction)
print('The accuracy using the standardized data with 5 principal components and \n' +
      '5 nearest neighbors is ' + str(acc2) + '. The file demonstrating this \n' +
      'output is titled \'Standardized PCA Results.csv\'.')
results.to_csv('Standardized PCA Results.csv', index=False)
```

```
The accuracy using the standardized data with 5 principal components and
5 nearest neighbors is 0.9433962264150944. The file demonstrating this
output is titled 'Standardized PCA Results.csv'.
```

```
C:\Users\epurn\PycharmProjects\CSC522\venv\lib\site-packages\sklearn\base.py:443: UserWarning: X has feature names, but KNeighborsClassifier was fitted without feature names
  warnings.warn(
```

In [32]:
```
# C.iv.b) Plot results with varying number of principal components.
points2 = plot_results(std_trainDF, std_testDF, traintarget, testtarget)
print(points2)
points2.plot(x='n Components', y='Accuracy')
plt.title('Accuracy vs. # of Principal Components - Standardized')
plt.show()
```

```
   n Components  Accuracy
0             1  0.679245
1             2  0.981132
2             3  0.943396
3             5  0.943396
4            10  0.943396
5            13  0.924528
```



In [33]:
```
# C.iv.c) This time it appears that the most reasonable choice is 2 principal
# components. Scaling likely provides a better representation of variance between
# the data points, as the data is centered and brought to a similar magnitude
# rather than being condensed to a domain of [0,1]. Normalization could possibly
# enhance outliers whereas scaling does not give too much extra weight to data
# points at the edges of the domain.
```

In [34]:
```
# D) Of the two results between normalized and standardized data, I would prefer
# to use the standardized data as it is able to give a comparable best accuracy
```

```
# (around 98%) to the normalized data using 2 principal components instead of 5
# principal components as with the normalized data.
```

In [ ]:

In [ ]: