In [1]:

```
import sklearn.datasets
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_wine
from sklearn.datasets import load_digits
from sklearn.preprocessing import MinMaxScaler
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import mglearn as mglearn
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn import linear_model
from sklearn.decomposition import PCA
from numpy import linalg as LA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_predict
from sklearn.manifold import TSNE
```

In [2]:

```
#                                                    Question 1
print("Question 1")
print()
print()
print()
```

```
Question 1
```

In [3]:

```
wine = load_wine()
df = pd.DataFrame(wine.data,columns=wine.feature_names)
df['target'] = pd.Series(wine.target)
X_train, X_test, y_train, y_test = train_test_split(wine.data,
                                                    wine.target,
                                                    stratify = w
ine.target,
                                                    random_state
= 42)
df.head()
print()
print(wine.target_names)
```

```
['class_0' 'class_1' 'class_2']
```

In [4]:

```
wine.DESCR
```

Out[4]:

```
'Wine Data Database\n==================\n\nNotes\n
-----\nData Set Characteristics:\n    :Number of Ins
tances: 178 (50 in each of three classes)\n    :Numb
er of Attributes: 13 numeric, predictive attributes
and the class\n    :Attribute Information:\n \t\t- 1
) Alcohol\n \t\t- 2) Malic acid\n \t\t- 3) Ash\n\t\t
- 4) Alcalinity of ash  \n \t\t- 5) Magnesium\n\t\t-
6) Total phenols\n \t\t- 7) Flavanoids\n \t\t- 8) No
nflavanoid phenols\n \t\t- 9) Proanthocyanins\n\t\t-
10)Color intensity\n \t\t- 11)Hue\n \t\t- 12)OD280/O
D315 of diluted wines\n \t\t- 13)Proline\n        \t
- class:\n                - class_0\n
- class_1\n                - class_2\n\t\t\n    :Sum
mary Statistics:\n    \n    ========================
===== ==== ===== ======= =====\n
Min   Max   Mean     SD\n    ========================
====== ==== ===== ======= =====\n    Alcohol:
11.0  14.8    13.0   0.8\n    Malic Acid:
0.74  5.80    2.34  1.12\n    Ash:
1.36  3.23    2.36  0.27\n    Alcalinity of Ash:
10.6  30.0    19.5   3.3\n    Magnesium:
70.0 162.0    99.7  14.3\n    Total Phenols:
```

0.98 3.88   2.29 0.63\n   Flavanoids:
0.34 5.08   2.03 1.00\n   Nonflavanoid Phenols:
0.13 0.66   0.36 0.12\n   Proanthocyanins:
0.41 3.58   1.59 0.57\n   Colour Intensity:
1.3 13.0   5.1   2.3\n   Hue:
0.48 1.71   0.96 0.23\n   OD280/OD315 of diluted
wines: 1.27 4.00   2.61 0.71\n   Proline:
278 1680   746   315\n   ======================
====== ==== ===== ======= =====\n\n   :Missing Attr
ibute Values: None\n   :Class Distribution: class_0
(59), class_1 (71), class_2 (48)\n   :Creator: R.A.
Fisher\n   :Donor: Michael Marshall (MARSHALL%PLU@i
o.arc.nasa.gov)\n   :Date: July, 1988\n\nThis is a
copy of UCI ML Wine recognition datasets.\nhttps://a
rchive.ics.uci.edu/ml/machine-learning-databases/win
e/wine.data\n\nThe data is the results of a chemical
analysis of wines grown in the same\nregion in Italy
by three different cultivators. There are thirteen d
ifferent\nmeasurements taken for different constitue
nts found in the three types of\nwine.\n\nOriginal O
wners: \n\nForina, M. et al, PARVUS - \nAn Extendibl
e Package for Data Exploration, Classification and C
orrelation. \nInstitute of Pharmaceutical and Food A
nalysis and Technologies,\nVia Brigata Salerno, 1614
7 Genoa, Italy.\n\nCitation:\n\nLichman, M. (2013).
UCI Machine Learning Repository\n[http://archive.ics
.uci.edu/ml]. Irvine, CA: University of California,\
nSchool of Information and Computer Science. \n\nRef
erences\n----------\n(1) \nS. Aeberhard, D. Coomans
and O. de Vel, \nComparison of Classifiers in High D
imensional Settings, \nTech. Rep. no. 92-02, (1992),
Dept. of Computer Science and Dept. of \nMathematics
and Statistics, James Cook University of North Queen
sland. \n(Also submitted to Technometrics). \n\nThe
data was used with many others for comparing various
\nclassifiers. The classes are separable, though onl
y RDA \nhas achieved 100% correct classification. \n
(RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-tran
sformed data)) \n(All results using the leave-one-ou
t technique) \n\n(2) \nS. Aeberhard, D. Coomans and
O. de Vel, \n"THE CLASSIFICATION PERFORMANCE OF RDA"
\nTech. Rep. no. 92-01, (1992), Dept. of Computer Sc
ience and Dept. of \nMathematics and Statistics, Jam
es Cook University of North Queensland. \n(Also subm

itted to Journal of Chemometrics). \n'

In [5]:

```
print( X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(133, 13) (45, 13) (133,) (45,)
```

In [6]:

```
#Scaling Data#
print("Scaling Data")
print('')
scaler = MinMaxScaler()
scaler.fit(X_train)
MinMaxScaler(copy=True, feature_range=(0, 1))
X_train_scaled = scaler.transform(X_train)   # transform data
print("transformed shape: {}".format(X_train_scaled.shape))    #
print dataset properties before and after scaling
print('')
print("per-feature minimum before scaling:\n {}".format(X_train.
min(axis=0)))
print('')
print("per-feature maximum before scaling:\n {}".format(X_train.
max(axis=0)))
print('')
print("per-feature minimum after scaling:\n {}".format(X_train_s
caled.min(axis=0)))
print('')
print("per-feature maximum after scaling:\n {}".format(X_train_s
caled.max(axis=0)))
```

```
Scaling Data

transformed shape: (133, 13)

per-feature minimum before scaling:
 [1.103e+01 7.400e-01 1.360e+00 1.060e+01 7.000e+01
9.800e-01 3.400e-01
 1.300e-01 4.200e-01 1.280e+00 4.800e-01 1.270e+00 2
.780e+02]

per-feature maximum before scaling:
 [1.483e+01 5.800e+00 3.220e+00 3.000e+01 1.620e+02
3.880e+00 3.740e+00
 6.300e-01 3.580e+00 1.300e+01 1.710e+00 3.920e+00 1
.515e+03]

per-feature minimum after scaling:
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

per-feature maximum after scaling:
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]


In [7]:
```

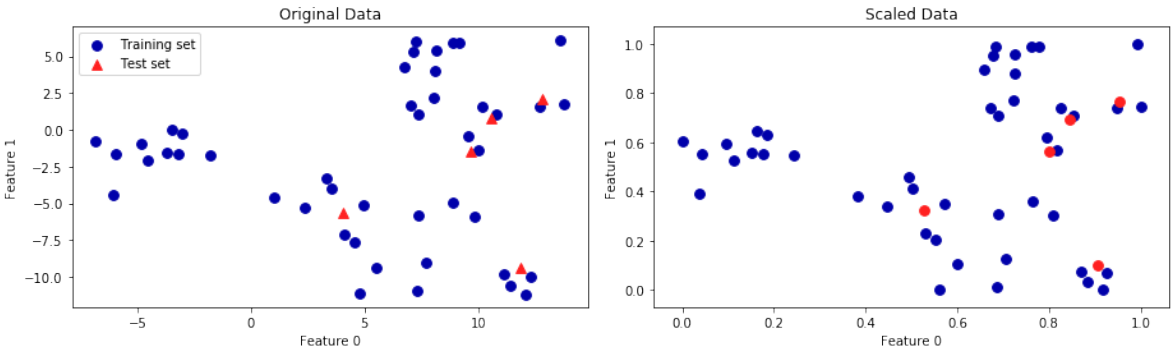
```
X_test_scaled = scaler.transform(X_test)
#print dataset properties before and after scaling
print("transformed shape: {}".format(X_test_scaled.shape))
print('')
print("per-feature minimum before scaling:\n {}".format(X_test.m
in(axis=0)))
print('')
print("per-feature maximum before scaling:\n {}".format(X_test.m
ax(axis=0)))
print('')
print("per-feature minimum after scaling:\n {}".format(X_test_sc
aled.min(axis=0)))
print('')
print("per-feature maximum after scaling:\n {}".format(X_test_sc
aled.max(axis=0)))
```

```
transformed shape: (45, 13)

per-feature minimum before scaling:
 [1.156e+01 8.900e-01 1.700e+00 1.200e+01 7.800e+01
1.150e+00 4.800e-01
 1.700e-01 4.100e-01 2.150e+00 5.700e-01 1.290e+00 2
.900e+02]

per-feature maximum before scaling:
 [1.438e+01 5.650e+00 3.230e+00 2.850e+01 1.280e+02
3.520e+00 5.080e+00
 6.600e-01 2.960e+00 1.068e+01 1.450e+00 4.000e+00 1
.680e+03]

per-feature minimum after scaling:
 [ 0.13947368  0.02964427  0.1827957   0.07216495  0
.08695652  0.05862069
  0.04117647  0.08       -0.00316456  0.07423208  0.
07317073  0.00754717
  0.00970089]

per-feature maximum after scaling:
 [0.88157895 0.97035573 1.00537634 0.92268041 0.6304
3478 0.87586207
 1.39411765 1.06       0.80379747 0.80204778 0.78861
789 1.03018868
 1.13338723]


In [8]:
```

```
X, _ = make_blobs(n_samples = 50, centers = 5, random_state = 4,
cluster_std = 2)
X_train, X_test = train_test_split(X, random_state = 5, test_siz
e = 0.1)
```

In [9]:

```
#plot training and test sets
fig, axes = plt.subplots(1, 2, figsize=(13, 4))
axes[0].scatter(X_train[:, 0], X_train[:, 1], c=mglearn.cm2(0),
label="Training set", s=60)
axes[0].scatter(X_test[:, 0], X_test[:, 1], marker='^', c=mglear
n.cm2(1), label='Test set', s=60)
axes[0].legend(loc='upper left')
axes[0].set_title("Original Data")

#scale data using MinMaxScalar
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
#visualize the properly scaled data
axes[1].scatter(X_train_scaled[:, 0], X_train_scaled[:, 1], c=mg
learn.cm2(0), label="Training set", s=60)
axes[1].scatter(X_test_scaled[:, 0], X_test_scaled[:, 1], c=mgle
arn.cm2(1), label="Test set", s=60)
axes[1].set_title("Scaled Data")

for ax in axes:
    ax.set_xlabel("Feature 0")
    ax.set_ylabel("Feature 1")
    fig.tight_layout()
```



In [10]:

```
X_train, X_test, y_train, y_test = train_test_split(wine.data, w
ine.target, stratify = wine.target, random_state=66)
svm = SVC(C=100)
svm.fit(X_train, y_train)
print("Test set accuracy: {:.2f}".format(svm.score(X_test, y_tes
t)))
print('')
print("Preprocessing using 0-1 scaling with MinMaxScaler")
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
print('')
print("Learn an SVM on scaled training data:")
svm.fit(X_train_scaled, y_train)
print("Scoring on scaled test set")
print("Scaled test set accuracy: {:.2f}".format(svm.score(X_test
_scaled, y_test)))
```

```
Test set accuracy: 0.49

Preprocessing using 0-1 scaling with MinMaxScaler

Learn an SVM on scaled training data:
Scoring on scaled test set
Scaled test set accuracy: 0.96
```

In [11]:

```
print("Test set accuracy: {:.2f}".format(svm.score(X_test, y_tes
t)))
print('')
print("Preprocessing using 0-1 scaling with MinMaxScaler")
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
print('')
svm.fit(X_train_scaled, y_train)
print("SVM test set accuracy: {:.2f}".format(svm.score(X_test_sc
aled, y_test)))
```

```
Test set accuracy: 0.40

Preprocessing using 0-1 scaling with MinMaxScaler

SVM test set accuracy: 0.98
```

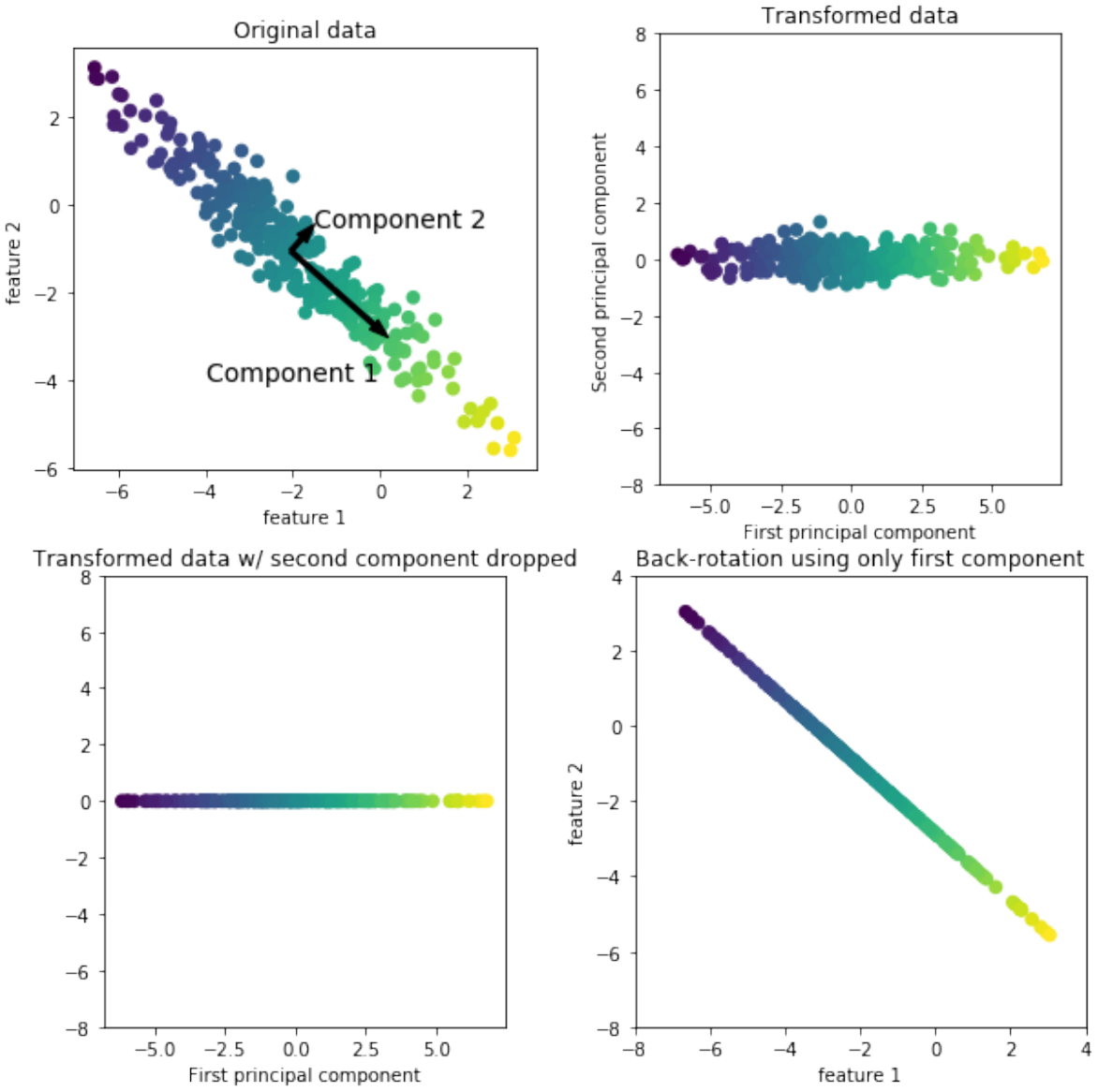
In [12]:

```
#                                                    Question 6
print("Question 2")
print()
print()
print()
```

```
Question 2
```

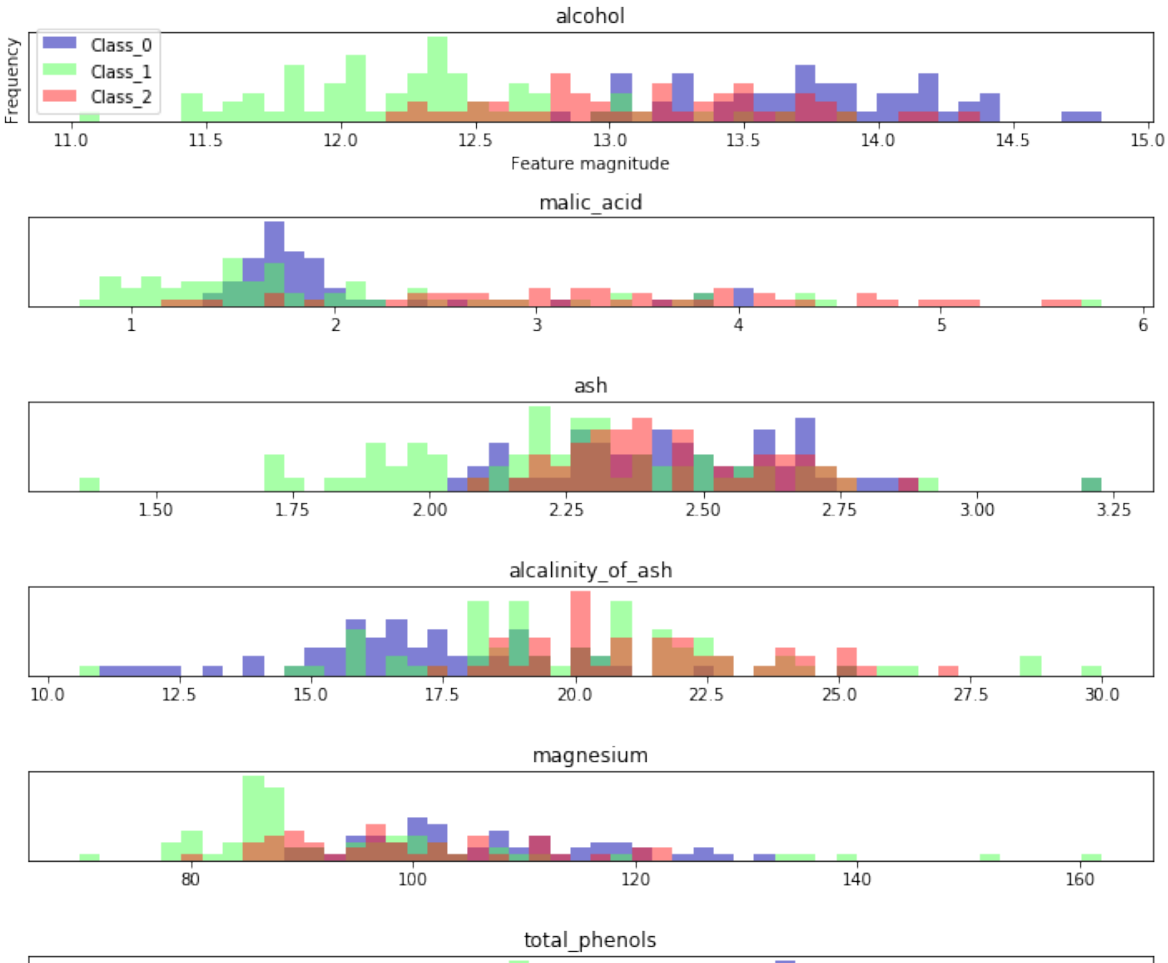
In [13]:

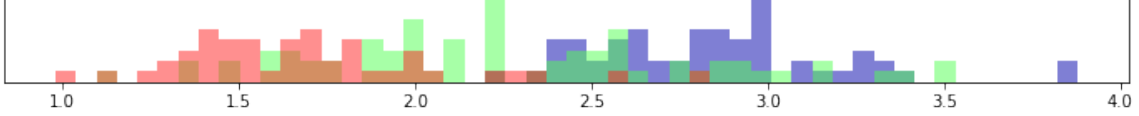
```
mglearn.plots.plot_pca_illustration()
```

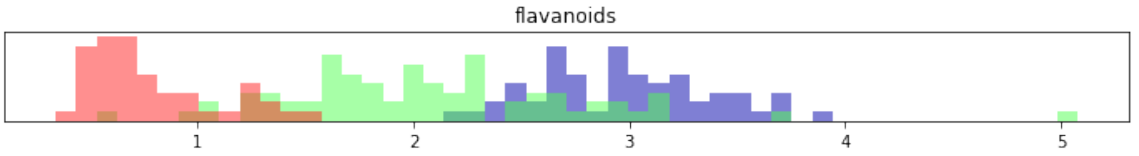


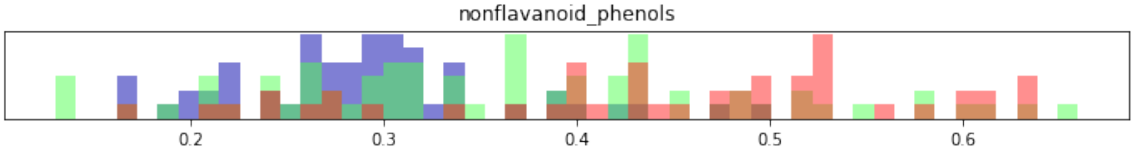
In [14]:

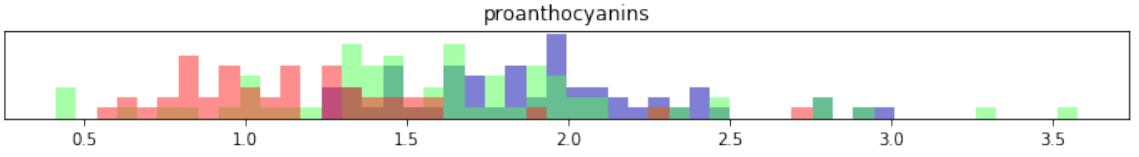
```
fig, axes = plt.subplots (13, 1, figsize = (10, 20))
Class_0 = wine.data[wine.target == 0]
Class_1 = wine.data [wine.target == 1]
Class_2 = wine.data [wine.target == 2]
ax = axes.ravel()
for i in range (13):
    _ , bins = np.histogram (wine.data[:,i], bins = 50)
    ax [i].hist(Class_0 [:,i], bins = bins, color = mglearn.cm3
(0), alpha = .5)
    ax [i].hist(Class_1 [:,i], bins = bins, color = mglearn.cm3
(2), alpha = .5)
    ax [i].hist(Class_2 [:,i], bins = bins, color = mglearn.cm3
(1), alpha = .5)
    ax [i].set_title(wine.feature_names [i])
    ax [i].set_yticks(())
    ax [0].set_xlabel( "Feature magnitude" )
    ax [0].set_ylabel( "Frequency" )
    ax [0].legend (["Class_0", "Class_1", "Class_2"], loc = "bes
t")
fig.tight_layout ()
```

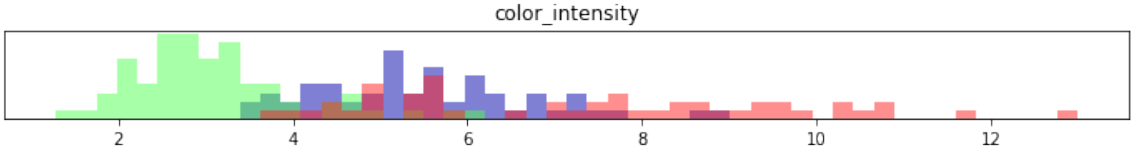


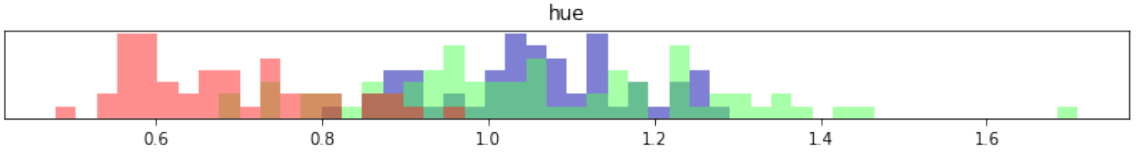

1.0
1.5
2.0
2.5
3.0
3.5
4.0

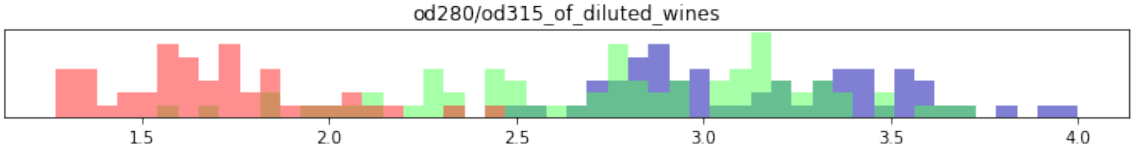

flavanoids
1
2
3
4
5

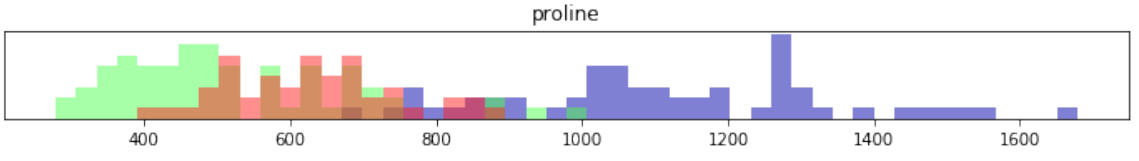

nonflavanoid_phenols
0.2
0.3
0.4
0.5
0.6


proanthocyanins
0.5
1.0
1.5
2.0
2.5
3.0
3.5


color_intensity
2
4
6
8
10
12


hue
0.6
0.8
1.0
1.2
1.4
1.6

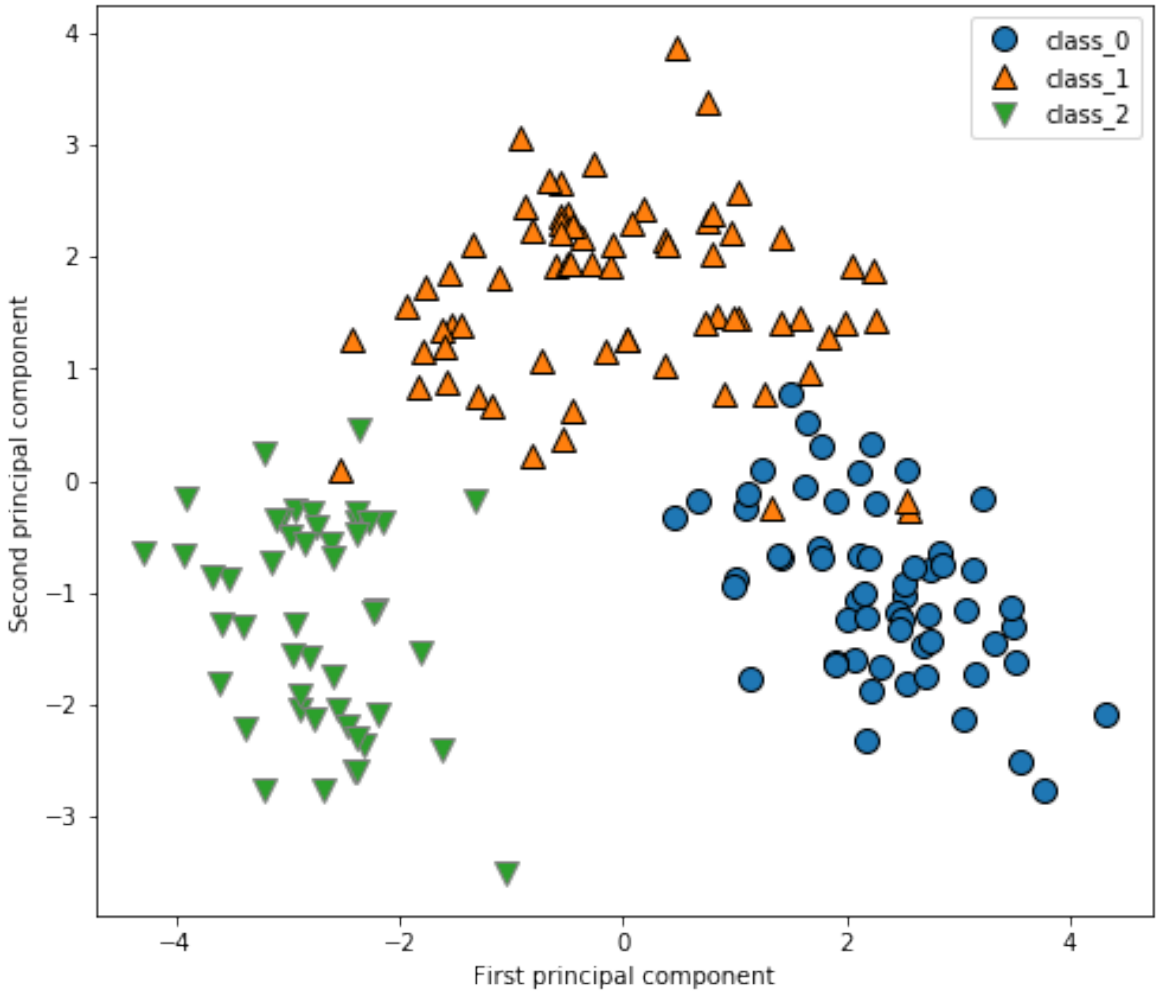

od280/od315_of_diluted_wines
1.5
2.0
2.5
3.0
3.5
4.0


proline
400
600
800
1000
1200
1400
1600

In [15]:

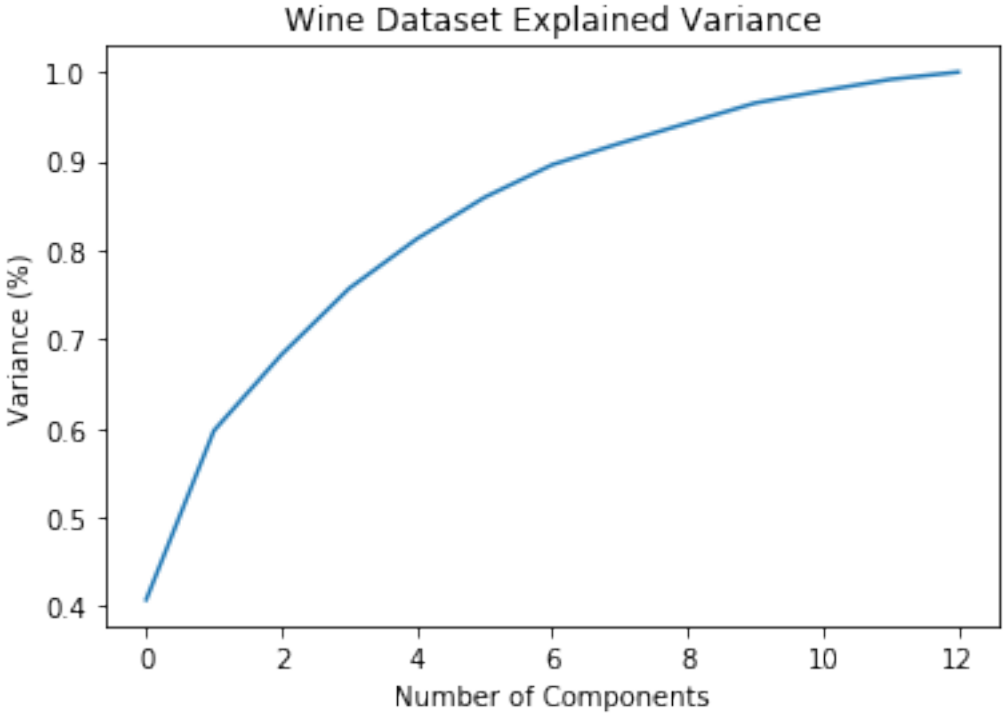
```
print("Scale Dataset")
scaler = StandardScaler()
scaler.fit(wine.data)
X_scaled = scaler.transform(wine.data)
print('')

pca = PCA(n_components = 4)
pca.fit(X_scaled)

#transform data onto the first two principal components
X_pca = pca.transform(X_scaled)
print("Original shape: {}".format(str(X_scaled.shape)))
print("Reduced shape: {}".format(str(X_pca.shape)))
print('')

#plot first vs. second principal components
plt.figure(figsize=(8, 8))
mglearn.discrete_scatter(X_pca[:, 0], X_pca[:, 1], wine.target)
plt.legend(wine.target_names, loc="best")
plt.gca().set_aspect("equal")
plt.xlabel("First principal component")
plt.ylabel("Second principal component")
```

```
Scale Dataset

Original shape: (178, 13)
Reduced shape: (178, 4)


Out[15]:

Text(0,0.5,'Second principal component')
```



In [16]:

```
scaler = MinMaxScaler(feature_range=[0, 1])
wine_rescaled = scaler.fit_transform(X_scaled)

#Fitting the PCA algorithm with our Data
pca = PCA().fit(wine_rescaled)

#Plotting the Cumulative Summation of the Explained Variance
plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Variance (%)') #for each component
plt.title('Wine Dataset Explained Variance')
plt.show()
```

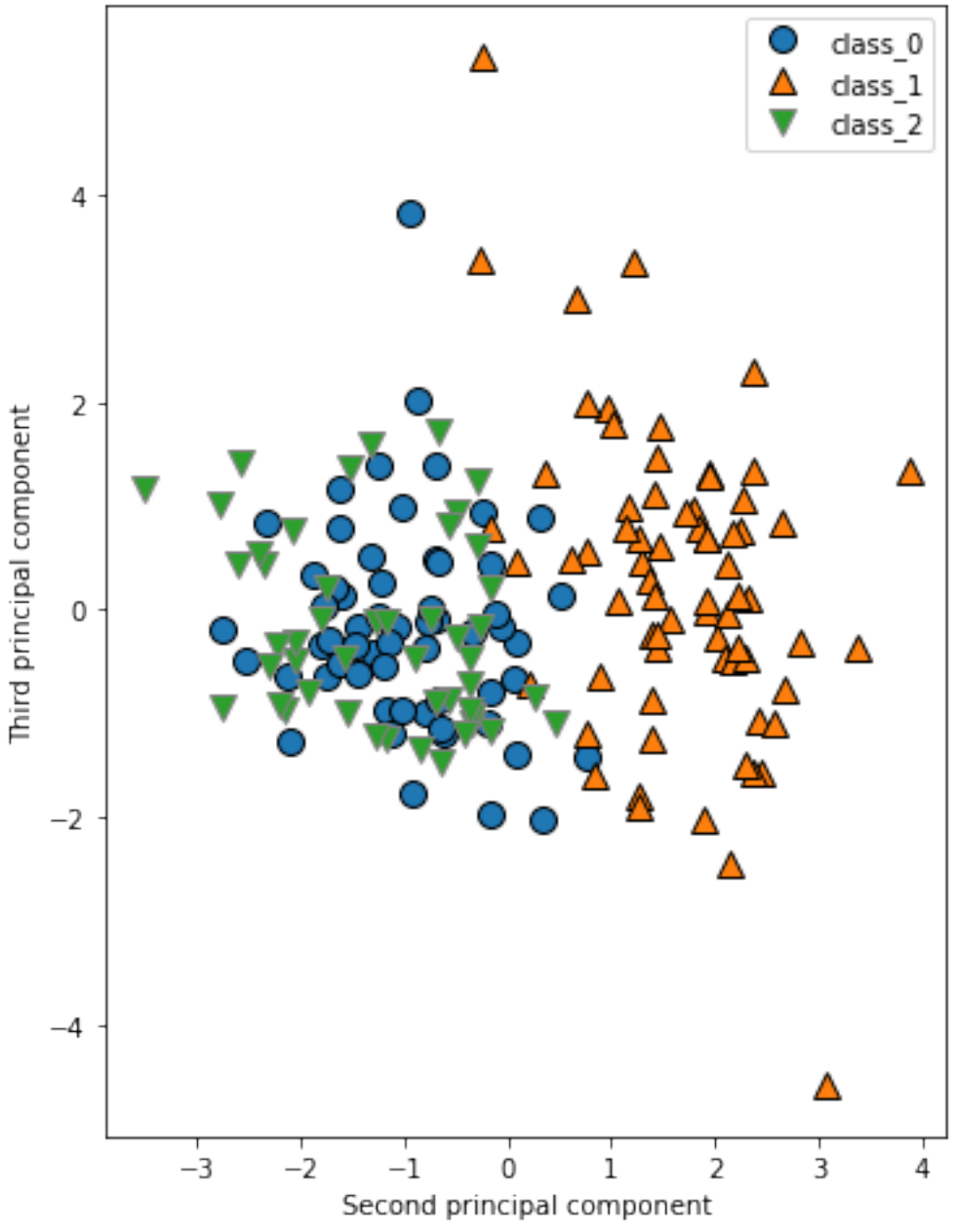


In [17]:

```
#plot second vs. third principal components
plt.figure(figsize=(8, 8))
mglearn.discrete_scatter(X_pca[:, 1], X_pca[:, 2], wine.target)
plt.legend(wine.target_names, loc="best")
plt.gca().set_aspect("equal")
plt.xlabel("Second principal component")
plt.ylabel("Third principal component")
```

Out[17]:

```
Text(0,0.5,'Third principal component')
```

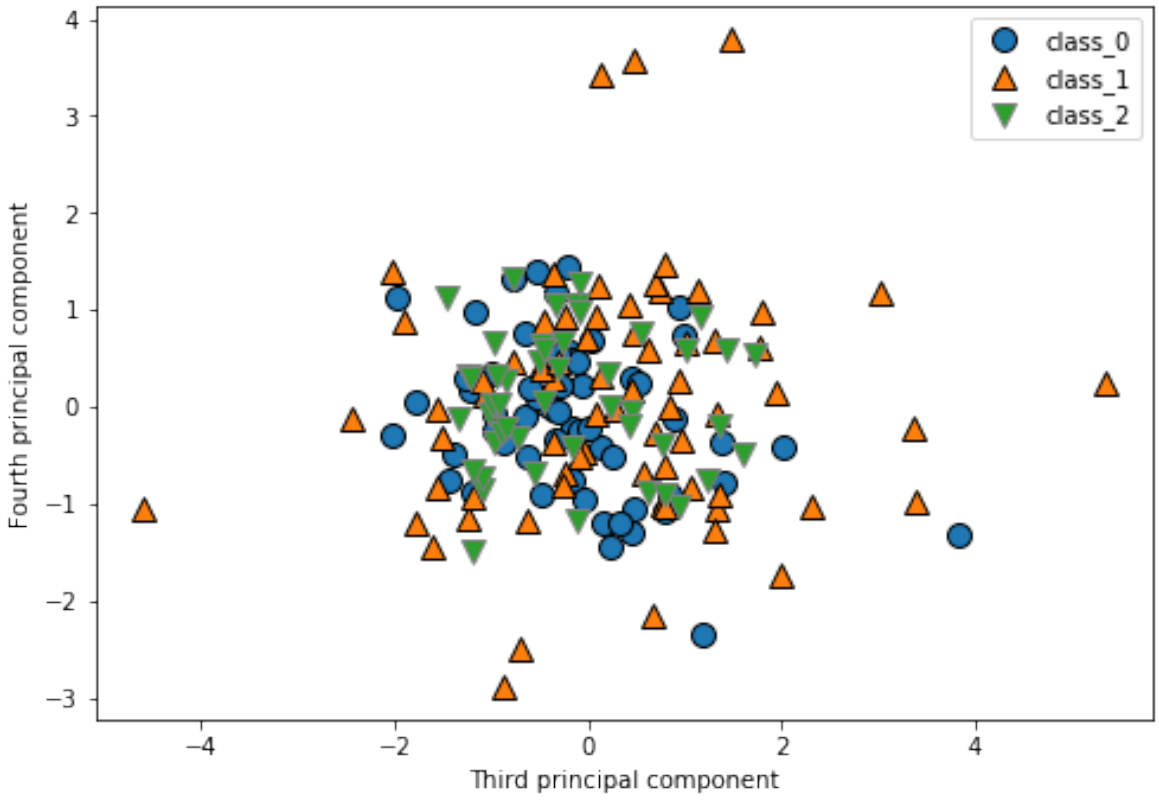


In [18]:

```
#plot secind vs. third principal components, colored by class
plt.figure(figsize=(8, 8))
mglearn.discrete_scatter(X_pca[:, 2], X_pca[:, 3], wine.target)
plt.legend(wine.target_names, loc="best")
plt.gca().set_aspect("equal")
plt.xlabel("Third principal component")
plt.ylabel("Fourth principal component")
```

Out[18]:

```
Text(0,0.5,'Fourth principal component')
```



In [19]:

```
print("PCA component shape: {}".format(pca.components_.shape))
print("PCA components:\n{}".format(pca.components_))

plt.matshow(pca.components_, cmap='viridis')
plt.yticks([0, 1, 2, 3], ["first component", "Second component",
"Third Component", "Third Component"])
plt.colorbar()
plt.xticks(range(len(wine.feature_names)), wine.feature_names, r
otation=60, ha='left')
plt.xlabel("Feature")
plt.ylabel("Principal components")
```

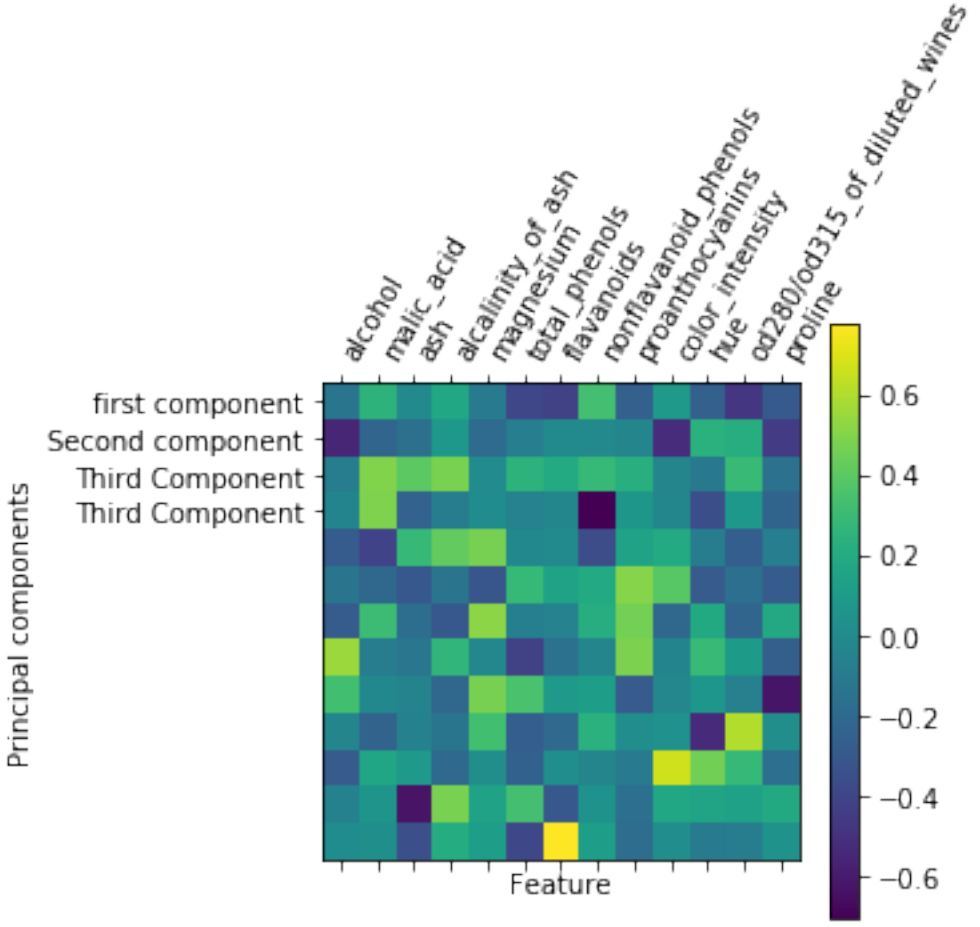
```
PCA component shape: (13, 13)
PCA components:
[[-1.33367664e-01  2.48515807e-01 -7.39167565e-04  1
.77838621e-01
  -8.86572802e-02 -3.95070868e-01 -4.14589792e-01  3
.33108614e-01
  -2.52902105e-01  9.23290406e-02 -2.51137258e-01 -4
.73492101e-01
  -2.86862112e-01]
 [-5.50883679e-01 -2.27390577e-01 -1.63091200e-01  7
.97763293e-02
  -1.88165658e-01 -7.41447292e-02 -1.00692215e-03 -9
.96036899e-03
  -3.14178847e-02 -5.19707496e-01  2.37206223e-01  2
.15562246e-01
  -4.43888361e-01]
 [-8.38483848e-02  4.92039271e-01  4.03009367e-01  4
.77242411e-01
   6.55069647e-03  2.53065140e-01  1.96105456e-01  2
.85981862e-01
   2.28342345e-01 -3.30999741e-02 -1.06588781e-01  2
.97776289e-01
  -1.51968292e-01]
 [-4.03281504e-02  4.86032016e-01 -2.41975652e-01 -8
.16939916e-02
   1.58801742e-02 -5.26432313e-02 -2.70332340e-02 -7
.09095281e-01
   7.65736687e-02 -2.63220354e-02 -3.52498013e-01  8
.20620051e-02
  -2.29792503e-01]
```

```
 [-2.78010089e-01 -4.12889958e-01  2.86393800e-01  4
.18304878e-01
   4.70898721e-01 -1.80297818e-02  2.57297682e-03 -3
.60037857e-01
   1.48665877e-01  2.00582427e-01 -7.71620754e-02 -2
.63396482e-01
  -7.65558029e-02]
 [-1.37254317e-01 -2.13102785e-01 -3.00964675e-01 -1
.35609774e-01
  -3.12518646e-01  2.83604411e-01  1.47044786e-01  1
.97231755e-01
   5.03115302e-01  3.88031900e-01 -2.83041419e-01 -1
.67278047e-01
  -2.76348795e-01]
 [-2.75352205e-01  3.10301653e-01 -1.71805330e-01 -2
.97555954e-01
   5.20419359e-01 -6.65236704e-02 -5.06982802e-02  2
.14888676e-01
   4.66206579e-01 -2.12595824e-01  1.93314973e-01 -2
.24561421e-01
   1.88372489e-01]
 [ 5.55030622e-01 -7.69281456e-02 -1.23460354e-01  2
.70756271e-01
  -1.99366070e-02 -4.13383802e-01 -1.56460745e-01 -3
.42321550e-02
   4.83470145e-01 -3.69237057e-02  2.95371717e-01  1
.02303314e-01
  -2.60228843e-01]
 [ 3.26686683e-01 -1.44706000e-02 -4.43615384e-02 -2
.04923518e-01
   4.71910296e-01  3.51231855e-01  8.89602892e-02  1
.22889237e-01
  -2.84054087e-01 -2.37079985e-02  6.87746576e-02 -6
.01970243e-02
  -6.24838818e-01]
 [-3.59116648e-02 -2.37072825e-01 -5.68986375e-02 -1
.32503040e-01
   3.25867728e-01 -2.63006307e-01 -1.99444858e-01  2
.36774685e-01
   1.87429817e-02  3.91434937e-02 -5.34248137e-01  6
.02193751e-01
   2.38290504e-02]
 [-2.80684930e-01  1.70374648e-01  9.07566359e-02 -2
.00864181e-01
   2.72773854e-02 -2.47764000e-01  3.05792695e-02 -3
```

```
.25501002e-02
  -9.93148337e-02   6.69144086e-01   4.58587544e-01   2
.87815292e-01
  -1.69476208e-01]
 [-5.61269360e-02   6.16542748e-02 -6.32537780e-01   4
.76650308e-01
   1.44634964e-01   3.33709022e-01 -2.99870770e-01   4
.73112536e-02
  -1.64956850e-01   1.79211241e-01   1.57379260e-01   1
.41063254e-01
   1.93164791e-01]
 [ 1.13596663e-02   2.94277481e-02 -3.48604449e-01   2
.12755282e-01
   1.23445378e-01 -3.87627848e-01   7.75691511e-01   1
.18318269e-01
  -1.80607204e-01   1.62431688e-02 -9.56238231e-02 -8
.17454885e-02
   4.81186875e-02]]
```

Out[19]:

```
Text(0,0.5,'Principal components')
```

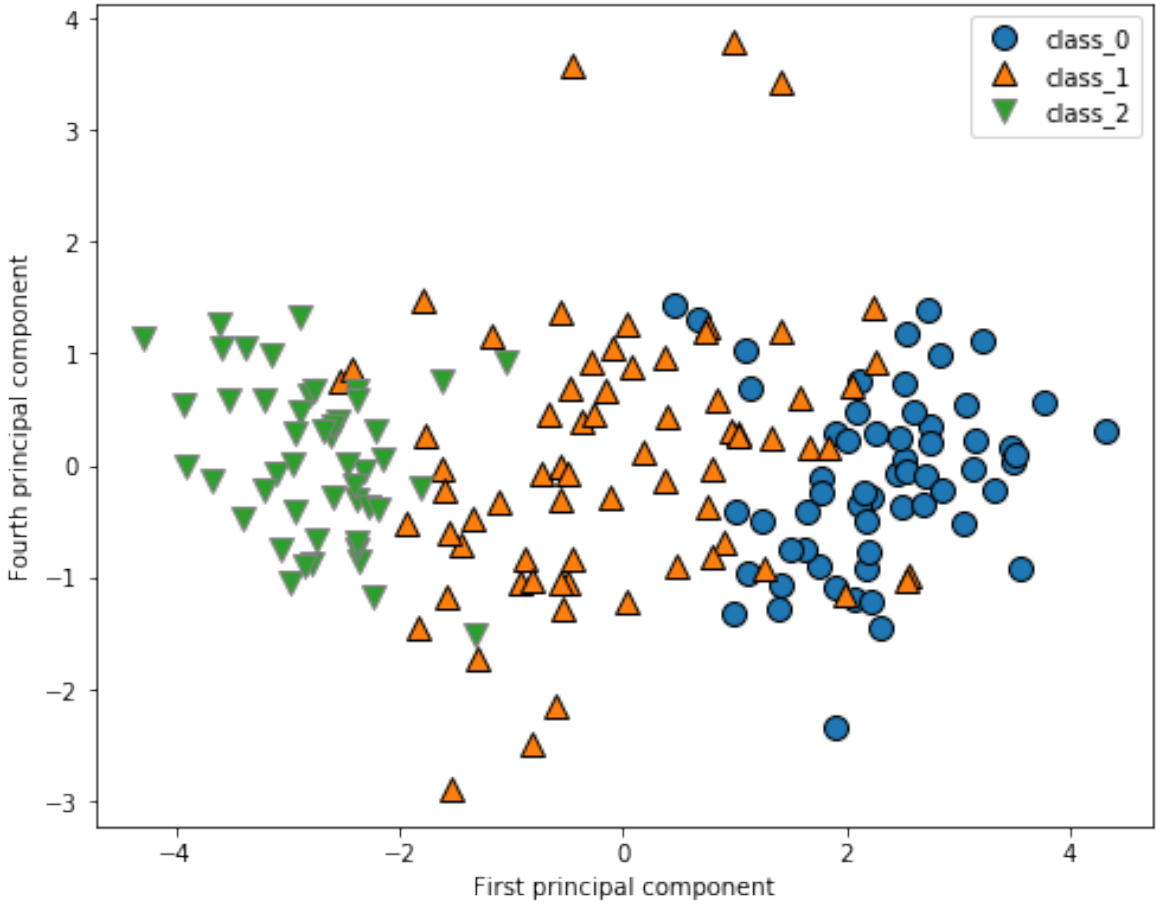


In [20]:

```
plt.figure(figsize=(8, 8))
mglearn.discrete_scatter(X_pca[:, 0], X_pca[:, 3], wine.target)
plt.legend(wine.target_names, loc="best")
plt.gca().set_aspect("equal")
plt.xlabel("First principal component")
plt.ylabel("Fourth principal component")
```

Out[20]:

```
Text(0,0.5,'Fourth principal component')
```



In [21]:

```
counts = np.bincount(wine.target)
for i, (count, name) in enumerate(zip(counts, wine.target_names)
):
    print("{0:8} {1:10}".format(name, count), end="  |  ")
    if (i+1) % 3 == 0:
        print()
```

```
class_0          59  |  class_1          71  |  clas
s_2          48  |
```

In [22]:

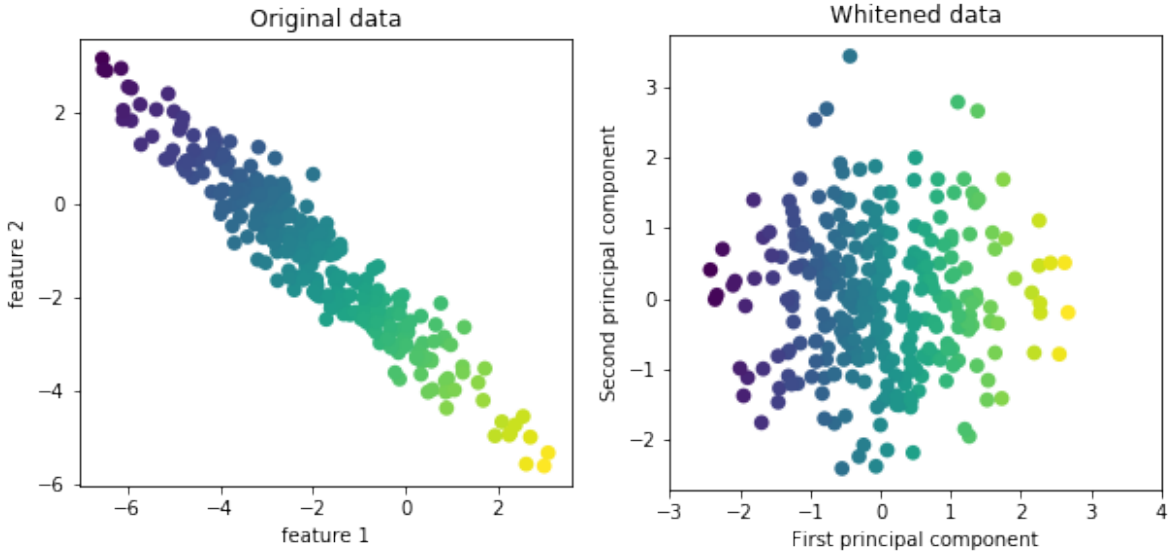
```
mask = np.zeros(wine.target.shape, dtype=np.bool)
for target in np.unique(wine.target):
    mask[np.where(wine.target == target)[0][:50]] = 1

X_wine = wine.data[mask]
y_wine = wine.target[mask]

X_train, X_test, y_train, y_test = train_test_split(X_wine, y_wi
ne, stratify=y_wine, random_state=0)

knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
print()
print("Test set score of 1-nn: {:.2f}".format(knn.score(X_test,
y_test)))
```

```
Test set score of 1-nn: 0.76
```

In [23]:

```
mglearn.plots.plot_pca_whitening()

pca = PCA(n_components=4, whiten=True, random_state=0).fit(X_tra
in)
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

print("X_train_pca.shape: {}".format(X_train_pca.shape))
X_train_pca.shape: (1547, 100)
```

```
X_train_pca.shape: (111, 4)
```



In [24]:

```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train_pca, y_train)
print()
print("Test set score of 1-nn: {:.2f}".format(knn.score(X_test_p
ca, y_test)))
```

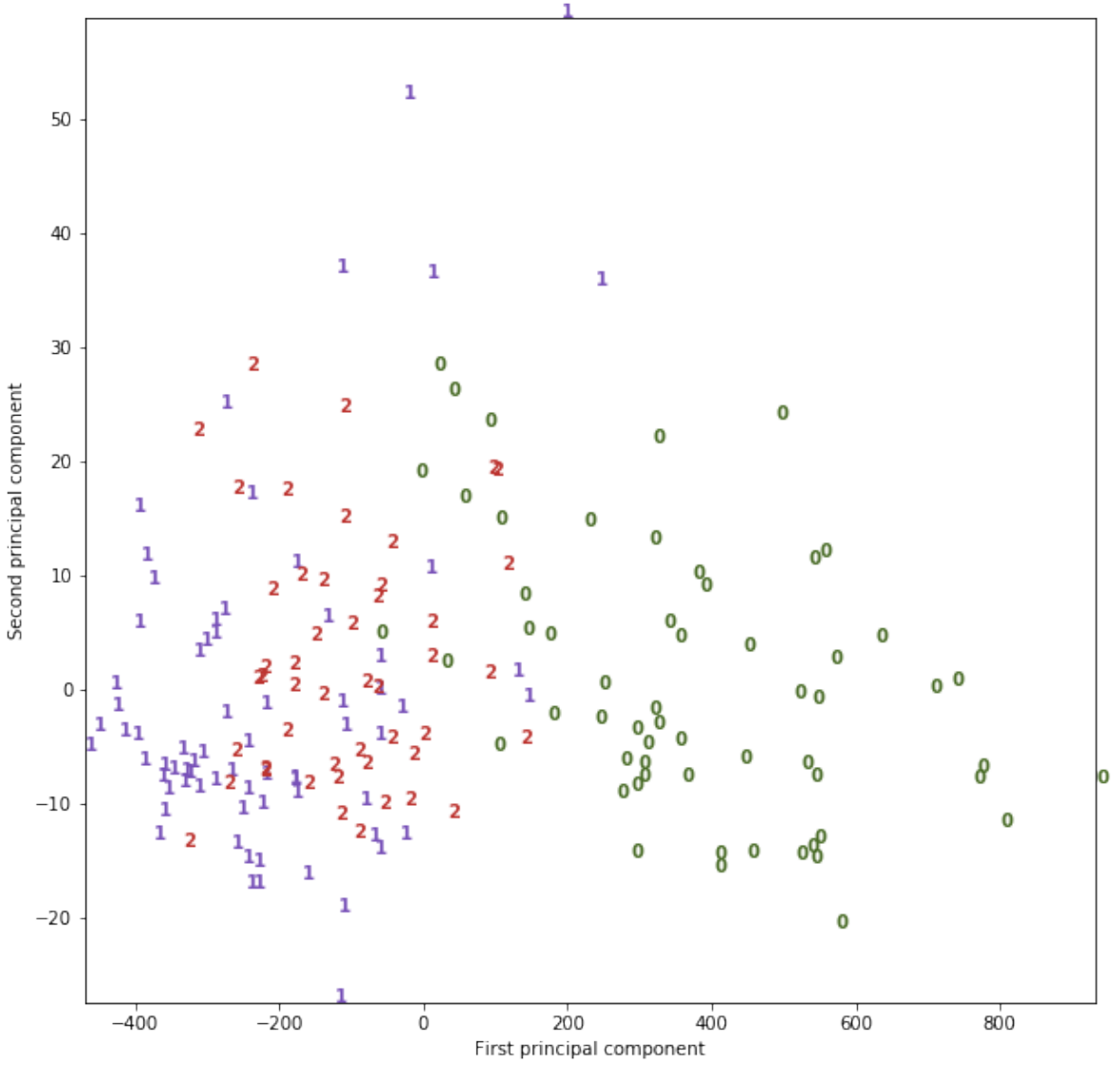
```
Test set score of 1-nn: 0.89
```

In [25]:

```
#build a PCA model
pca = PCA(n_components=4)
pca.fit(wine.data)

#transform the digits data onto the first two principal componen
ts
wine_pca = pca.transform(wine.data)
plt.figure(figsize=(10, 10))
plt.xlim(wine_pca[:, 0].min(), wine_pca[:, 0].max())
plt.ylim(wine_pca[:, 1].min(), wine_pca[:, 1].max())
colors = ["#476A24", "#7851B8", "#BD3430"]
for i in range(len(wine.data)):
    plt.text(wine_pca[i, 0], wine_pca[i, 1], str(wine.target[i])
, color = colors[wine.target[i]], fontdict={'weight': 'bold', 's
ize': 10})
plt.xlabel("First principal component")
plt.ylabel("Second principal component")
```

Out[25]:

Text(0,0.5,'Second principal component')

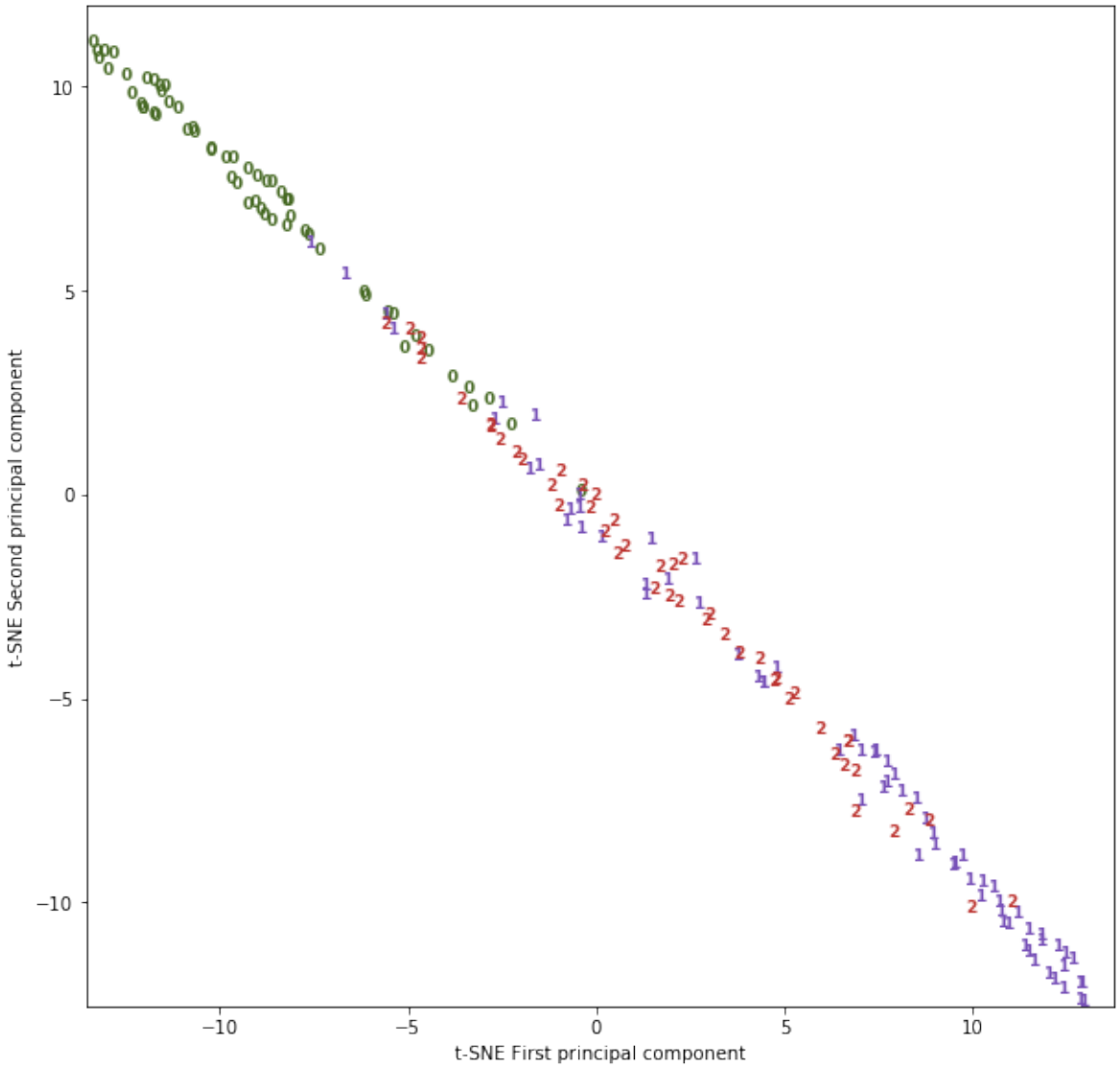


In [26]:

```
tsne = TSNE(random_state=42)

#use fit_transform instead of fit, as TSNE has no transform meth
od
wine_tsne = tsne.fit_transform(wine.data)
plt.figure(figsize=(10, 10))
plt.xlim(wine_tsne[:, 0].min(), wine_tsne[:, 0].max()+1)
plt.ylim(wine_tsne[:, 1].min(), wine_tsne[:, 1].max()+1)
for i in range(len(wine.data)):

    plt.text(wine_tsne[i, 0], wine_tsne[i, 1], str(wine.target[i
]), color = colors[wine.target[i]], fontdict={'weight': 'bold',
'size': 9})
plt.xlabel("t-SNE First principal component")
plt.ylabel("t-SNE Second principal component")
```

Out[26]:

Text(0,0.5,'t-SNE Second principal component')



In [27]:

```
#                                                    Question 6
print("Question 6")
print()
print()
print()
```

Question 6

In [28]:

```
data = make_blobs(n_samples = 300, n_features = 2, centers = 3,
cluster_std = 1.5, random_state=20)
points = data[0]
print(data)
```

```
(array([[  1.78609651,   9.31126487],
       [ -0.49335071,   9.9252263 ],
       [  6.84849892,   6.57299724],
       [  1.94369933,   8.5753122 ],
       [ -8.98023097,   5.72146774],
       [  7.97575621,   7.9479754 ],
       [  1.69632382,  10.30572339],
       [  8.04581521,   8.46901669],
       [  1.21502986,   8.69942993],
       [ -8.51804356,   2.45098271],
       [ -8.75185096,   5.01821248],
       [ -8.3999632 ,   2.35857999],
       [  7.37199215,   8.8851737 ],
       [  4.53768184,   7.69056487],
       [  1.14825888,  10.12538513],
       [  1.97804997,   9.07972145],
       [ -0.68163626,   5.23139085],
       [  9.74608435,   6.83644154],
       [ -8.94445614,   2.19648337],
       [  3.0250498 ,   8.33160389],
       [-11.26783907,   0.40555438],
       [  1.89284493,   4.18124314],
       [ -9.23504668,   2.10405052],
       [  2.97111988,   7.93544815],
       [-11.92676243,   5.34609493],
       [  9.91210855,   8.62307863],
       [ -9.72863006,   4.06559845],
       [  8.22870188,   9.18911701],
       [  8.53699625,   8.78797715],
       [  1.50964687,   3.16972037],
       [  1.37138166,   6.62005796],
       [  7.11431556,   6.13344583],
       [ -0.50915043,   7.1146326 ],
       [  0.09050509,   6.56946352],
       [  3.44281441,   9.95344187],
       [  7.22032772,   4.46992963],
       [  1.81990472,   9.4385044 ],
```

```
[  6.58329867,   5.30762964],
[  1.77946303,   6.32013036],
[  3.04825675,  10.17599003],
[  7.78542543,   7.26926916],
[  2.891086  ,   5.95189341],
[ -8.73066362,   3.09399534],
[  4.06908198,   6.40682578],
[  2.77757553,   9.20093323],
[-10.43458611,   6.60232846],
[  9.82686022,   4.78749305],
[ -8.78538274,   2.39548987],
[ 10.53212419,   6.02512131],
[ -9.8234493 ,   3.2487646 ],
[  0.31929204,   7.23059819],
[  1.41957012,   5.58989224],
[ -8.78073724,   6.65728296],
[  6.48315334,   7.41955281],
[-11.85874302,   3.63270157],
[  3.61855541,   9.21485753],
[  7.06401982,   7.20748356],
[  8.47696564,   8.52922908],
[  6.66950234,   4.71036594],
[  1.39760787,   7.75922849],
[  6.22766641,   7.65282563],
[ -8.01771719,   3.02757824],
[  8.680574  ,   5.1521588 ],
[ -0.46096155,   6.67002823],
[  7.48309485,   5.94531071],
[-10.16805417,   3.59883819],
[ -8.6264563 ,   2.80205577],
[  2.01136554,   9.13746978],
[ -9.67602005,   4.04507626],
[  6.70031754,   6.45098957],
[  5.47551164,   7.79571969],
[  6.35676092,   4.25629501],
[  7.42875348,   4.88940563],
[ -6.48257362,   0.86710693],
[  6.71531494,   4.93069168],
[  4.50332831,   6.3245189 ],
[ 10.89069221,   6.94079069],
[ -8.7366531 ,   2.31202468],
[ -8.22579998,   2.60369019],
[ -8.89408878,   3.53337988],
[  0.92983733,   7.9565374 ],
[  0.57459084,   6.69072266],
```

```
[  2.67749272,   7.83843293],
[-12.14827042,   4.50329204],
[ -0.91663693,   7.78678861],
[  4.17160498,   7.72378181],
[  6.84414397,   5.97630416],
[  8.00945964,   4.89983223],
[  1.70084581,   7.54878421],
[ -8.89668942,   4.97728157],
[ -8.30866861,   5.58921129],
[ -7.42250808,   4.57223736],
[  3.92483287,   8.3575229 ],
[  9.41366118,   7.98968894],
[ -8.4004175 ,   6.17407581],
[ -8.83882873,   3.21674813],
[  7.55341268,   9.60526125],
[  6.39258471,   5.81625252],
[ -8.05138119,   4.29288533],
[  0.17836252,   8.78568696],
[  6.49640182,   6.847011  ],
[  9.03705672,   8.41883694],
[ -1.14928342,   8.46642662],
[ -9.60535645,   2.94967962],
[  6.56733403,   4.59448753],
[-10.42503312,   2.2906264 ],
[  3.17182005,   6.486553  ],
[  8.64297327,   6.09908859],
[-10.88991761,   6.07267152],
[  4.90461403,   8.49341805],
[ -8.83431256,   4.40577561],
[ -9.82973895,   3.3821686 ],
[  9.24425885,   7.63648335],
[  1.9914848 ,   9.79918896],
[  9.55041228,   7.10733384],
[ 10.11777441,   8.12693086],
[-11.54568026,   4.65363503],
[  8.83578928,   4.59344051],
[  0.55226439,   6.89480903],
[  2.51726128,   8.56389626],
[  0.69493769,  10.11889907],
[  9.54181323,   6.51138205],
[  0.36020908,   5.53478923],
[  4.55689872,   7.51142885],
[  2.41965277,   9.99095808],
[  2.17073135,   8.06688557],
[-11.46873792,   2.52177954],
```

```
[-10.25615974,   5.63374877],
[ -9.39937964,   2.73719899],
[  5.70483041,   7.81745935],
[-10.41764085,   2.55249655],
[-11.04775055,   3.13983289],
[  3.40576605,   7.45395888],
[-10.27233821,   3.84868202],
[  6.74725697,   9.49931884],
[  6.36441629,   7.69935409],
[  7.06832734,   5.5739259 ],
[  0.43437629,   6.29740533],
[  9.58477008,   5.90302724],
[-10.28925267,   2.48839546],
[  3.35027916,   9.0750222 ],
[ -9.93166842,   2.73929835],
[  2.48009078,   6.88148881],
[  1.43312434,  10.20937179],
[  1.40341139,   8.6438909 ],
[  7.27275278,   4.75762353],
[  2.26980554,   8.11786873],
[ -8.99643854,   5.9843712 ],
[-11.26308092,   2.99612728],
[-10.48964959,   3.41124725],
[  7.02146845,   7.20344866],
[  8.58426744,   8.05969842],
[ -8.17503077,   6.00002366],
[ -0.6612328 ,   6.67632765],
[ -9.21177027,   3.16654626],
[  5.51661797,   7.82287026],
[  6.88156254,   5.77745952],
[-11.72098821,   4.15254045],
[  7.54056182,   6.85121859],
[ 10.01578127,   4.7461171 ],
[  7.47955003,   5.9336901 ],
[ -0.40015684,   5.82927444],
[-10.56323162,   6.28630092],
[  2.19695475,   7.24896054],
[  8.56837547,   7.82231243],
[  7.65317435,   5.77604009],
[  9.95235134,   3.90205353],
[ -8.48190873,   4.27458615],
[  6.08601543,   7.45641261],
[  0.48045279,   7.1845569 ],
[  7.26532688,   6.12403817],
[  8.38771601,   6.97612766],
```

```
[-10.51238918,   3.04885024],
[  2.5532206 ,   5.62762346],
[  2.50277188,   3.89772669],
[-10.66884663,   3.14909247],
[  4.06294353,   5.69084182],
[  7.38545157,   8.10080139],
[  3.71476012,   6.4620717 ],
[  7.7304752 ,   4.91579179],
[  7.43225396,   4.75585989],
[  7.80844604,   8.00641871],
[  8.35926294,   4.9640256 ],
[ -9.58340612,   3.03341616],
[  1.59908996,  10.28855422],
[ -9.24499936,   4.53512835],
[  6.29824049,   7.0251665 ],
[  6.89020858,   4.70070484],
[  9.70785836,   5.72955148],
[ -9.54554087,   5.49972175],
[ -9.43829447,   4.02387653],
[ -9.28666991,   3.6832684 ],
[  2.73316715,   8.84410038],
[  7.09481485,   8.08850585],
[ -9.10648291,   3.76406376],
[  8.18349199,   6.17790986],
[  9.46688181,   6.37213875],
[  3.33927904,   8.5638272 ],
[ -8.33437844,   1.69616431],
[-11.90324903,   1.51808571],
[ -9.39127543,   3.81487698],
[  7.25066613,   6.35110225],
[ -9.06388258,   5.84284142],
[  2.2989208 ,   4.4393817 ],
[-10.85028221,   2.4069133 ],
[ -9.14444637,   3.54278633],
[  9.19524962,   8.48653194],
[-10.85671912,   5.87831755],
[ -8.77635489,   4.19160503],
[  1.79466958,   8.84589771],
[ -7.43679102,   1.89457418],
[  7.3306632 ,   7.70929253],
[ -7.54745426,   4.94508574],
[  7.58416266,   6.90784333],
[  1.60389644,  10.96912698],
[ -0.39580979,   7.48737119],
[  6.51736467,   4.30306494],
```

```
[  7.65773331,   5.79677893],
[  9.4740623 ,   7.18591695],
[-10.92279573,   4.49155021],
[ -5.76591018,   4.52322275],
[-12.43858452,   4.75209594],
[ -8.30203995,   3.87232985],
[  3.40044594,   9.35616084],
[  9.50886972,   5.85405337],
[  2.7991192 ,   5.72569515],
[  9.21658182,   3.91467826],
[  8.1420971 ,   6.09285071],
[  3.11781052,   5.63653553],
[ -9.88886777,   5.26635288],
[-10.15507543,   4.9031894 ],
[  3.00039296,  10.68758271],
[  8.12020077,   5.04949894],
[ -7.35767514,   2.55336679],
[ -9.78874113,   5.76826536],
[  8.06435002,   7.15016665],
[  6.76842933,   8.43795877],
[  0.96449715,   9.64291284],
[  8.97329373,   3.84648393],
[  2.73226127,   8.685379  ],
[  6.25748787,   4.72756522],
[  7.85366755,   6.31578312],
[  1.25488011,   8.43573073],
[  3.3419539 ,   7.81406763],
[ -9.15026636,   3.67518637],
[-11.13605556,   5.08347644],
[ -1.18022234,   5.17621719],
[ -9.10919598,   5.50138939],
[-11.03865122,   5.4610802 ],
[  6.76179842,   6.80203194],
[  2.40619708,   8.60885716],
[ -8.52311872,   3.44476186],
[  9.36191213,   9.27463238],
[  1.15581499,   9.67766208],
[ -8.70895099,   2.73568422],
[  0.51957687,   8.12534596],
[  9.78787192,   7.29059181],
[ -9.44521325,   6.71609761],
[  6.7931783 ,   6.91005701],
[  5.81974858,   5.84986578],
[  8.78672684,   3.90260513],
[ -6.05939721,   4.44565773],
```

```
       [ -8.98838293,   3.98988339],
       [  2.24780754,   7.21415824],
       [  1.95578355,   4.85385147],
       [  4.90979979,   6.15491571],
       [  2.28534932,   8.62795178],
       [ -9.90021989,   5.52344687],
       [ -7.52820698,   3.7437793 ],
       [  2.25703603,   6.24948475],
       [  0.13536714,   8.79381899],
       [ -9.7027016 ,   5.37458192],
       [ -9.0751916 ,   2.25585278],
       [ -0.15663797,   8.32284731],
       [ -9.72776514,   5.28370791],
       [  0.35425029,   5.10207848],
       [  9.11150412,   6.30480965],
       [  3.16191554,  11.04403156],
       [  7.64714068,   6.6455039 ],
       [  2.0906026 ,   7.36495162],
       [  1.91079894,   7.61179301],
       [ -9.98740709,   6.75227289],
       [  1.10522573,  10.08502224],
       [ -9.94467704,   4.49474136],
       [  7.31046763,   7.70185773],
       [  2.55420873,   8.18803189],
       [  2.84880694,   7.6840356 ],
       [  4.72787434,   8.31706335],
       [ -8.88841489,   5.00813882],
       [  0.85915399,  10.2713671 ],
       [ -6.42457651,   5.26415733],
       [  3.56938293,   9.98196884],
       [  3.81237461,   5.90944458],
       [  0.33758901,   9.24127358],
       [  4.69576417,   7.36691143],
       [ -7.83429749,   1.34897074],
       [-11.59029087,   5.78081416],
       [ -8.51872484,   2.47992107],
       [  7.76022692,   4.12659028],
       [-10.55873909,   4.40921694]]), array([0, 0,
1, 0, 2, 1, 0, 1, 0, 2, 2, 2, 1, 0, 0, 0, 0, 1, 2, 0
, 2, 0,
       2, 0, 2, 1, 2, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0,
1, 0, 0, 1, 0, 2, 0,
       0, 2, 1, 2, 1, 2, 0, 0, 2, 1, 2, 0, 1, 1, 1,
0, 1, 2, 1, 0, 1, 2,
       2, 0, 2, 1, 1, 1, 1, 2, 1, 0, 1, 2, 2, 2, 0,
```

```
0, 0, 2, 0, 0, 1, 1,
        0, 2, 2, 2, 0, 1, 2, 2, 1, 1, 2, 0, 1, 1, 0,
2, 1, 2, 0, 1, 2, 0,
        2, 2, 1, 0, 1, 1, 2, 1, 0, 0, 0, 1, 0, 0, 0,
0, 2, 2, 2, 1, 2, 2,
        0, 2, 1, 1, 1, 0, 1, 2, 0, 2, 0, 0, 0, 1, 0,
2, 2, 2, 1, 1, 2, 0,
        2, 1, 1, 2, 1, 1, 1, 0, 2, 0, 1, 1, 1, 2, 1,
0, 1, 1, 2, 0, 0, 2,
        1, 1, 0, 1, 1, 1, 1, 2, 0, 2, 1, 1, 1, 2, 2,
2, 0, 1, 2, 1, 1, 0,
        2, 2, 2, 1, 2, 0, 2, 2, 1, 2, 2, 0, 2, 1, 2,
1, 0, 0, 1, 1, 1, 2,
        2, 2, 2, 0, 1, 0, 1, 1, 0, 2, 2, 0, 1, 2, 2,
1, 1, 0, 1, 0, 1, 1,
        0, 0, 2, 2, 0, 2, 2, 1, 0, 2, 1, 0, 2, 0, 1,
2, 1, 1, 1, 2, 2, 0,
        0, 1, 0, 2, 2, 0, 0, 2, 2, 0, 2, 0, 1, 0, 1,
0, 0, 2, 0, 2, 1, 0,
        0, 1, 2, 0, 2, 0, 0, 0, 0, 2, 2, 2, 1, 2]))
```

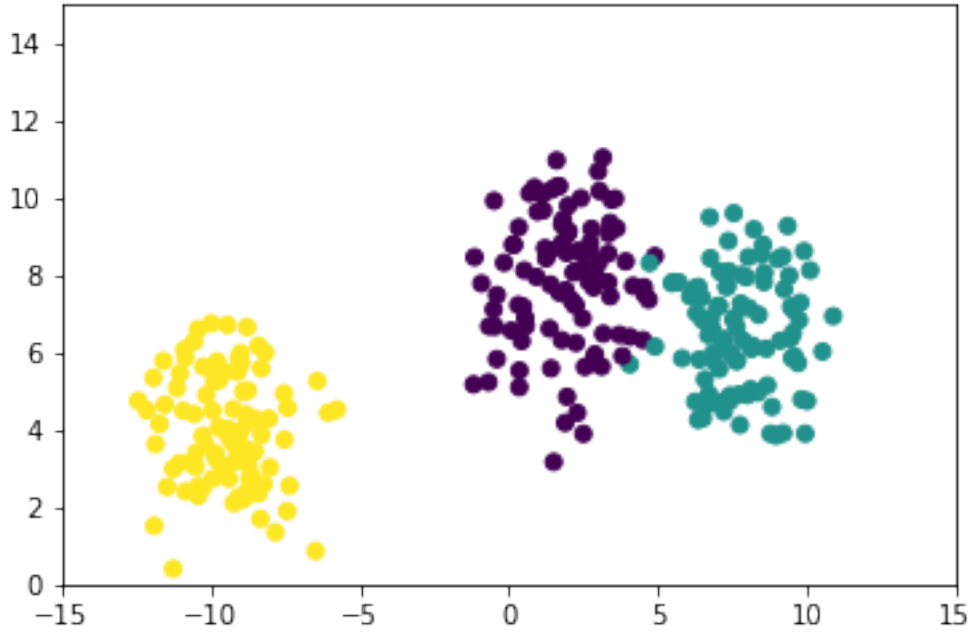
In [29]:

```
#Plotting data
print("Plotting data")
plt.scatter(data[0][:,0], data[0][:,1], c = data[1], cmap = 'vir
idis')
plt.xlim(-15, 15)
plt.ylim(0, 15)
```

Plotting data

Out[29]:

(0, 15)



In [30]:

```
#KMeans Clustering
print("KMeans Clustering")
kmeans = KMeans(n_clusters = 3)
kmeans.fit(points) #fit
print(kmeans.cluster_centers_) #cluster locations
y_km = kmeans.fit_predict(points) #saving clusters for charts
```

```
KMeans Clustering
[[ 7.84332115  6.52081551]
 [-9.45327176  3.98472662]
 [ 1.94132523  7.83718523]]
```

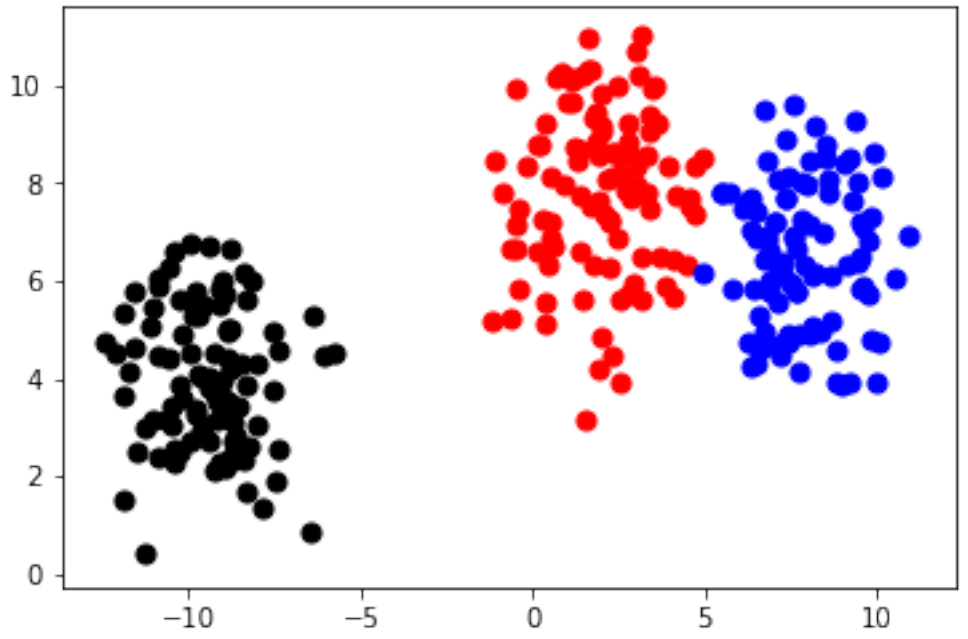
In [31]:

```
print("Plotting KMeans")
plt.scatter(points[y_km == 0,0], points[y_km == 0,1], s=50, c='r
ed')
plt.scatter(points[y_km == 1,0], points[y_km == 1,1], s=50, c='b
lack')
plt.scatter(points[y_km == 2,0], points[y_km == 2,1], s=50, c='b
lue')
```

```
Plotting KMeans
```

Out[31]:

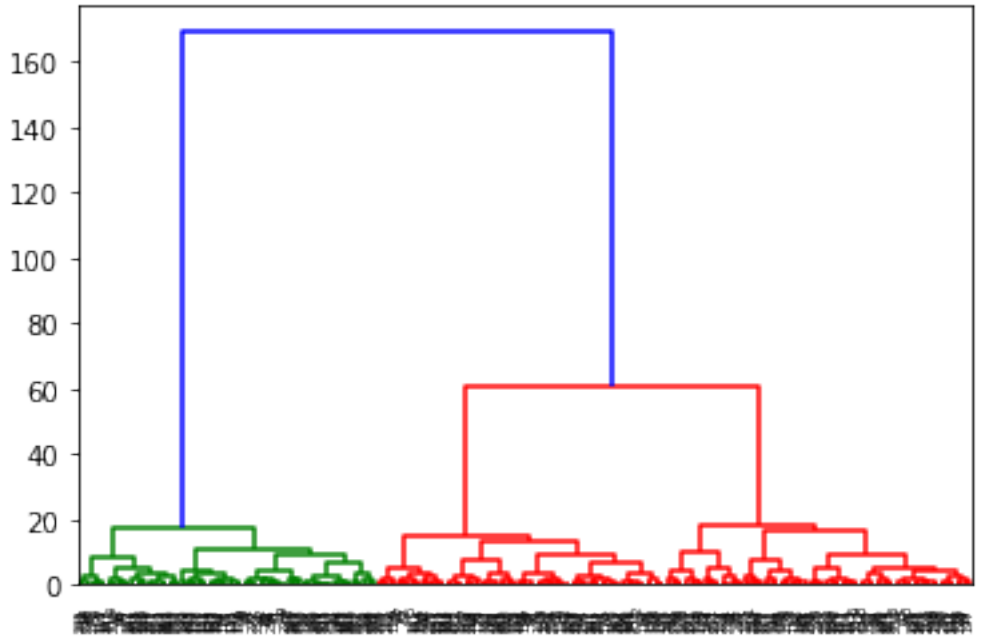
```
<matplotlib.collections.PathCollection at 0x1c23b2be
80>
```



In [32]:

```
#Hierarchical Clusering
print("Hierarchical Clusering")
dendrogram = sch.dendrogram(sch.linkage(points, method='ward'))
hc = AgglomerativeClustering(n_clusters=3, affinity = 'euclidean
', linkage = 'ward')
y_hc = hc.fit_predict(points)
```

```
Hierarchical Clusering
```



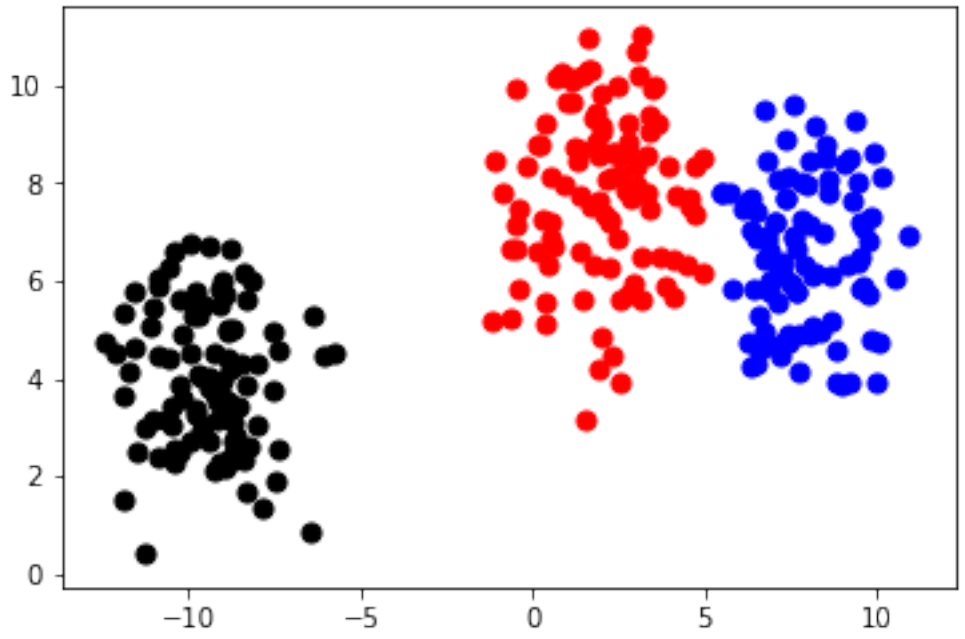
In [33]:

```
print("Plotting Hierarchical")
plt.scatter(points[y_hc ==0,0], points[y_hc == 0,1], s=50, c='red')
plt.scatter(points[y_hc==1,0], points[y_hc == 1,1], s=50, c='black')
plt.scatter(points[y_hc ==2,0], points[y_hc == 2,1], s=50, c='blue')
```

```
Plotting Hierarchical
```

Out[33]:

```
<matplotlib.collections.PathCollection at 0x1c23f009
40>
```



In [34]:

```
#                                                    Question 7
print("Question 7")
print()
print()
print()
```

```
Question 7
```

In [35]:

```
iris = load_iris()
iris.data.shape
```

Out[35]:

```
(150, 4)
```

In [36]:

```
X = iris.data[:, :2]
y = iris.target
print(X.shape)
print(y.shape)
```

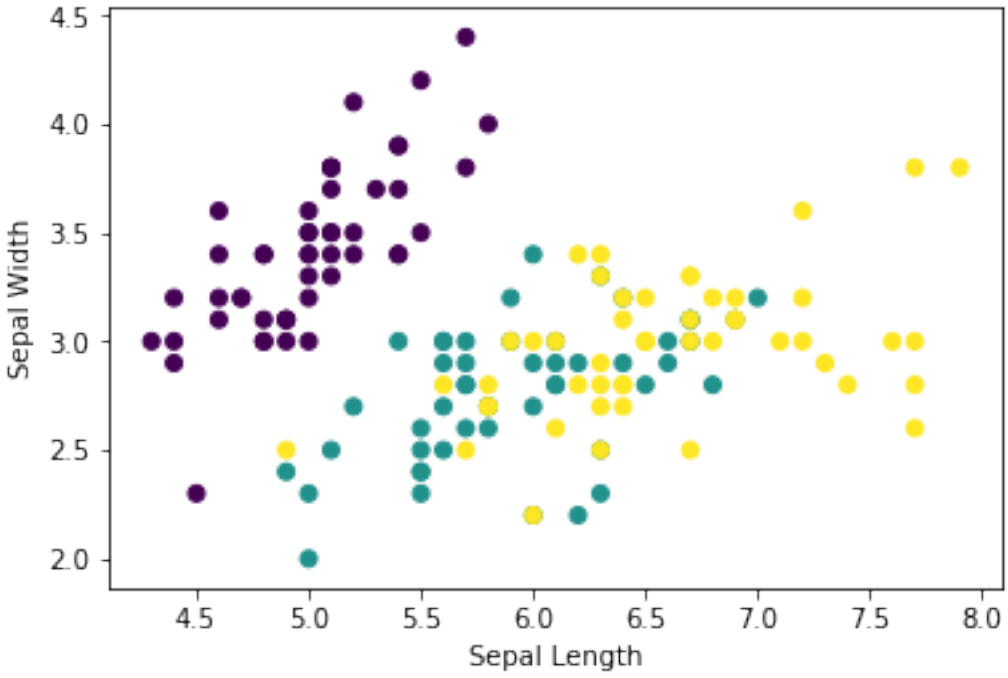
```
(150, 2)
(150,)
```

In [37]:

```
plt.scatter(X[:,0], X[:,1], c = y, cmap = 'viridis')
plt.xlabel('Sepal Length', fontsize=10)
plt.ylabel('Sepal Width', fontsize=10)
```

Out[37]:

```
Text(0,0.5,'Sepal Width')
```



In [48]:

```
kmeans = KMeans(n_clusters = 3, n_jobs = 4, random_state=21)
kmeans.fit(X) #fitting data
centers = kmeans.cluster_centers_
y_km = kmeans.fit_predict(points) #saving clusters for charts
print(centers)
```

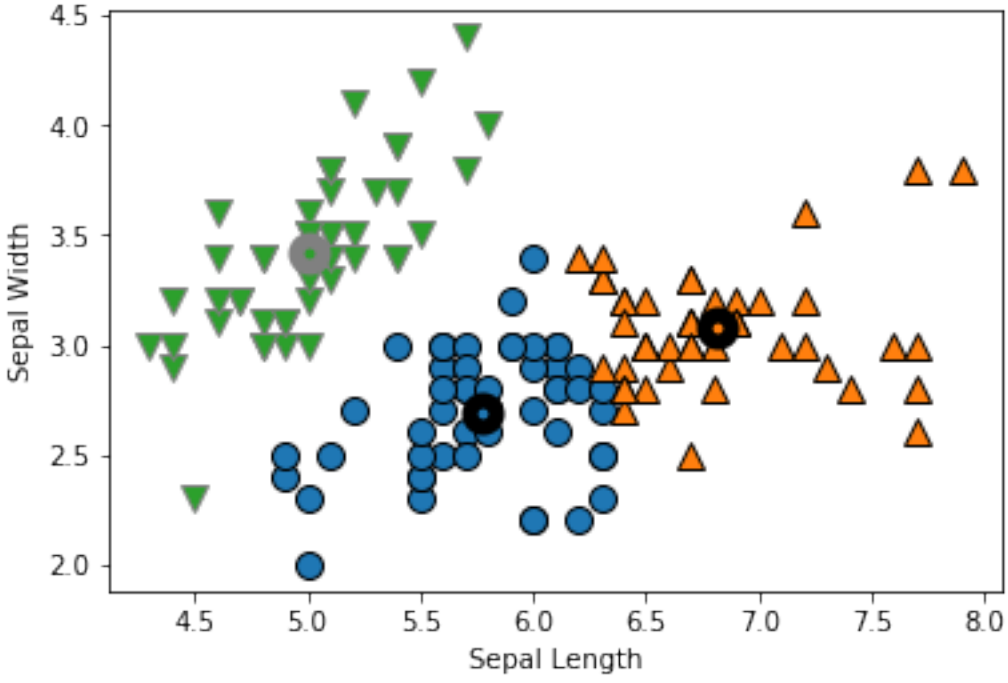
```
[[5.77358491 2.69245283]
 [5.006      3.418     ]
 [6.81276596 3.07446809]]
```

In [49]:

```
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)

mglearn.discrete_scatter(X[:, 0], X[:, 1], kmeans.labels_)
mglearn.discrete_scatter(kmeans.cluster_centers_[:, 0], kmeans.c
luster_centers_[:, 1], [0, 1, 2],
                         markers='o', markeredgewidth=6)
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
```

Out[49]:

```
Text(0,0.5,'Sepal Width')
```



In [41]:

```
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
print("Cluster:\n{}".format(kmeans.labels_))
print("k-Means Prediction:\n{}".format(kmeans.predict(X)))

lr = linear_model.LinearRegression()
y = iris.target

predicted = cross_val_predict(lr, iris.data, y, cv=5)

fig, ax = plt.subplots()
ax.scatter(y, predicted)
ax.plot([y.min(), y.max()], [y.min(), y.max()], ' ', lw=2)
ax.set_xlabel('Actual')
ax.set_ylabel('Predicted')
ax.set_title('Actual vs Predicted')
plt.show()
```

```
Cluster:
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 0 1 0 1 0 1 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0
 1 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 1 1 1 1 0 1 1 1 1
 1 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0 0 1 1 1 1 1 0 0 1 1
1 0 1 1 1 0 1 1 1 0 1
 1 0]
k-Means Prediction:
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 0 1 0 1 0 1 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0
 1 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 1 1 1 1 0 1 1 1 1
 1 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0 0 1 1 1 1 1 0 0 1 1
1 0 1 1 1 0 1 1 1 0 1
 1 0]
```