Exercise 5.1

The Python code for performing the PCA is given below in listing 1. Note that we set the parameter `random_state` for the `PCA` method to get reproducible results, in the case random numbers are used in the algorithm.

(a) Here, we explore histograms for the different variables in Fig. 1. None of the variables seem to separate the data into 3 distinct classes. However, some variables seem to separate the data partially, for instance, the variable flavanoids. Box plots[*] summarize

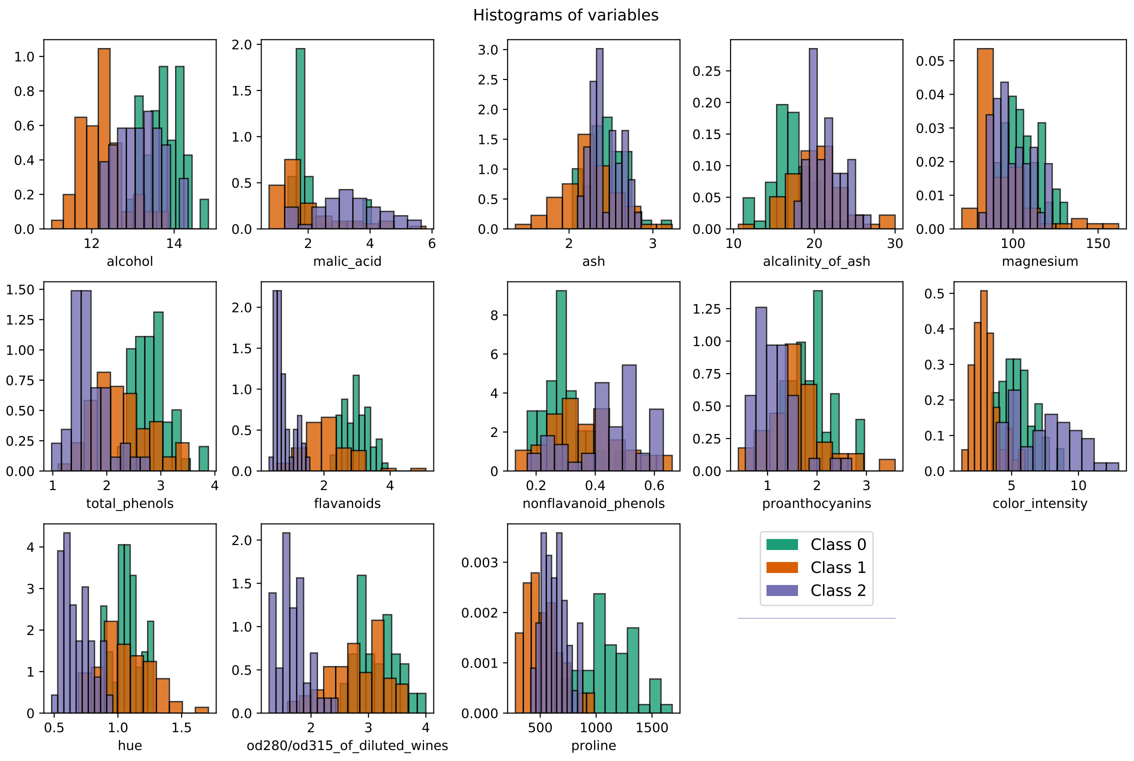

Figure 1: *Histograms of the variables in the wine data set.*

statistical data in a graphical way and box plots for the different variables are shown in Fig. 2. This figure also shows that there is no clear distinction between the classes. Some variables have distinct medians, but there is still some overlap within variables. Note that the box plots also indicate that there might be some outliers in some of the variables.

(b) Here, we should scale the variables. This ensures that we weight the different variables equally when we perform PCA. The explained variance is shown in Fig. 3. From this figure we see that we need 10 principal components to account for 95 % of the variance.

[*] https://www.youtube.com/watch?v=Hm6Mra5XJSs

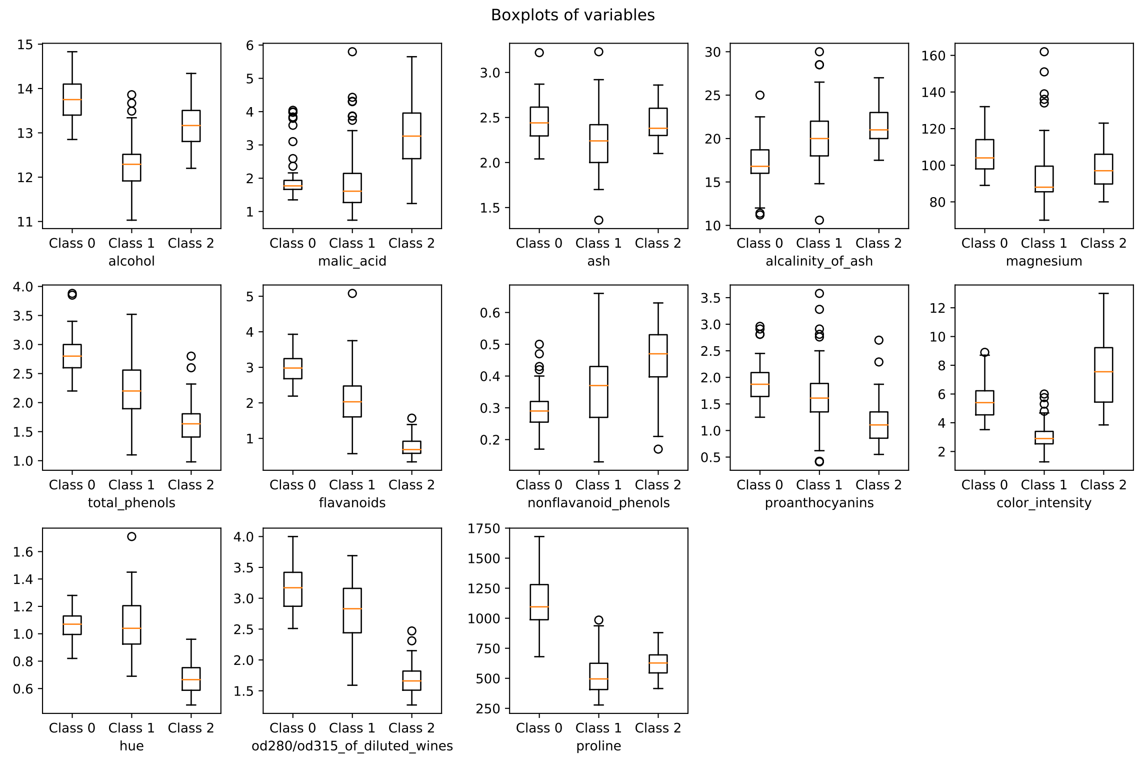

Figure 2: *Box plots for the variables in the wine data set.*

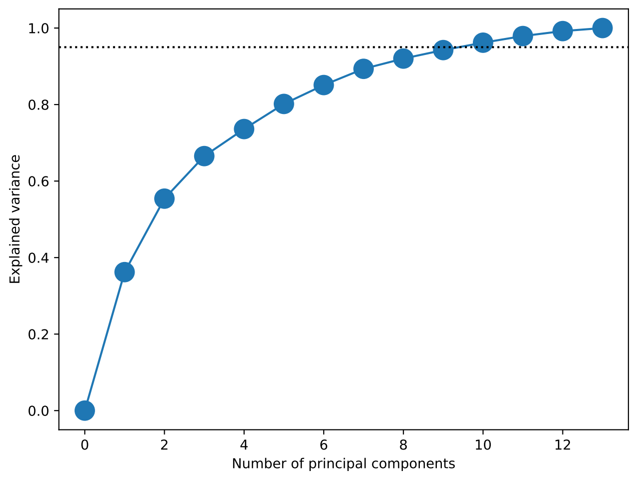

Figure 3: *Explained variance as a function of the number of principal components.*

(c) The variance obtained after running the PCA a second time is shown in Fig. 4. The scores for PC1 and PC2 are shown in Fig. 5. The scores show some separation into clusters, however, the distinction is not very clear.

(d) The loadings are shown in Fig. 6. From this figure, we see that flavanoids, total_phenols

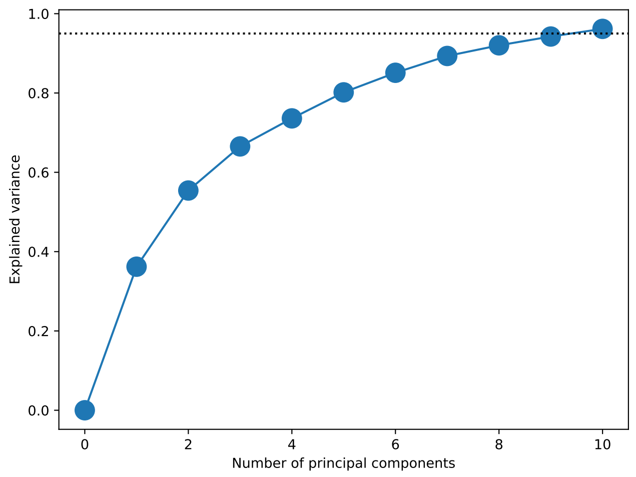

Figure 4: *Explained variance as a function of the number of principal components.*

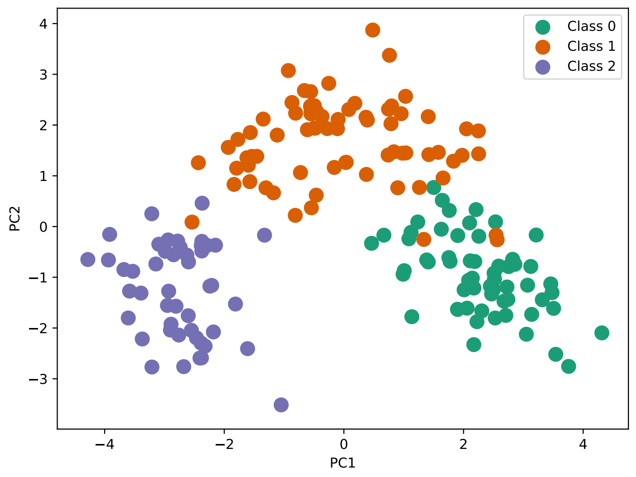

Figure 5: *Scores for PC1 and PC2.*

and proanthocyanins are grouped together and correlated.

(e) N/A.

Exercise 5.2

The Python code for performing the clustering is given below in listing 2. Note that we set the parameter `random_state` for the `KMeans` method to get reproducible results, in the case random numbers are used in the algorithm.

(a) The KMeans clustering starts with first defining the number of clusters we are going to find. Then we find initial positions for our centroids. This can, for instance, be done by

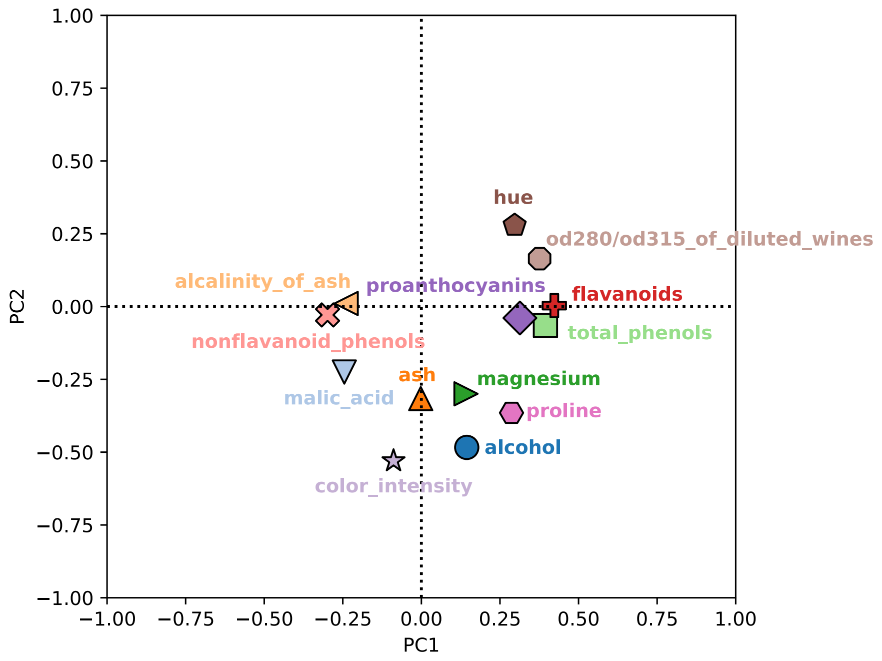

Figure 6: *Loadings for PC1 and PC2.*

giving the centroids random positions. In this specific case, we specify `init='k-means++'` (see listing 2) which is a bit more clever than just assigning random positions. In this initialization method, clusters are placed far away from each other. For the actual implementation please see the original paper describing this initialization.[†]

After having set the initial positions of the clusters, we do the following:

1 For each observation (data point) we assign it to the nearest centroid.

2 For each centroid, we calculate a new center by taking the mean of the locations of the observations assigned to it in the previous step.

3 We update the locations for the centeroids according to the mean found in the previous step.

4 We repeat the steps above until the locations of the cluster centers do not change.

If we do not know how many clusters there are, we have to try different values. After trying different possible cluster numbers, we compare them using metrics such as the sum of squared distances of the samples to their closest cluster center, or the silhouette values.

(b) The sum of squared distances of samples to their closest cluster center is given in Fig. 7. In this figure, there is no clear "elbow", however, we see that there is a large drop when

[†]http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf

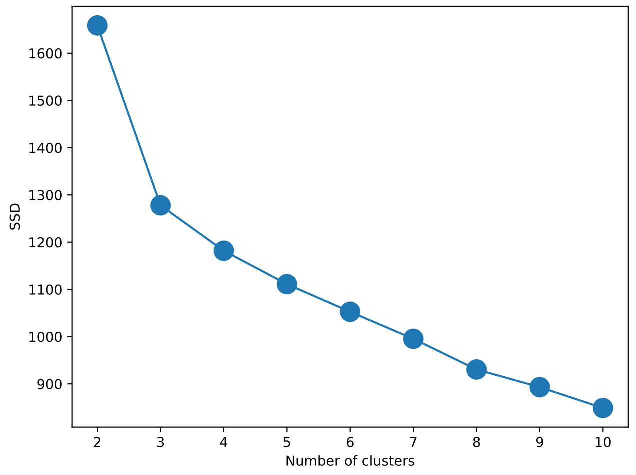

Figure 7: *Sum of squared distances (SSD) of samples to their closest cluster center.*

going from 2 to 3 clusters, and smaller drops when increasing the number of clusters. We can take this as an indication of there being three clusters in the data set, however, we should do some more analysis to see if we can verify this (see the next point).

(c) The silhouettes are given in Fig. 8 and the average values in Fig. 9. From these plots we see that using 2 or 3 clusters give more even clusterings than the other cluster numbers we tried. The average silhouette value is largest when using three clusters, and we find that there are three clusters.

We can also make some plots similar to the ones we made when exploring the raw data. A new box plot for the discovered clusters are shown in Fig. 10. Comparing this figure with Fig. 2 we see similarities (note that the cluster numbering differs), but here, we have discovered these clusters. We can, for instance, say that it looks like the color intensity of the different wines is something that can be used to distinguish between them.

(d) When repeating the clustering using the scores from the previous PCA, we get the results given in Fig. 11–14. In this case, we see that the clustering with three clusters stands out more compared to the situation where we considered all original variables. Here, we again conclude that there are three clusters in the data set.

Exercise 5.3 Example: LDA

Below you will find the results from the LDA example, see Fig. 15. Comparing with the results from the clustering, we see that the classes are better separated and that we can obtain regions defining the different classes.

The code for running LDA for the data we used in exercise 4 is given below in listing 3. The

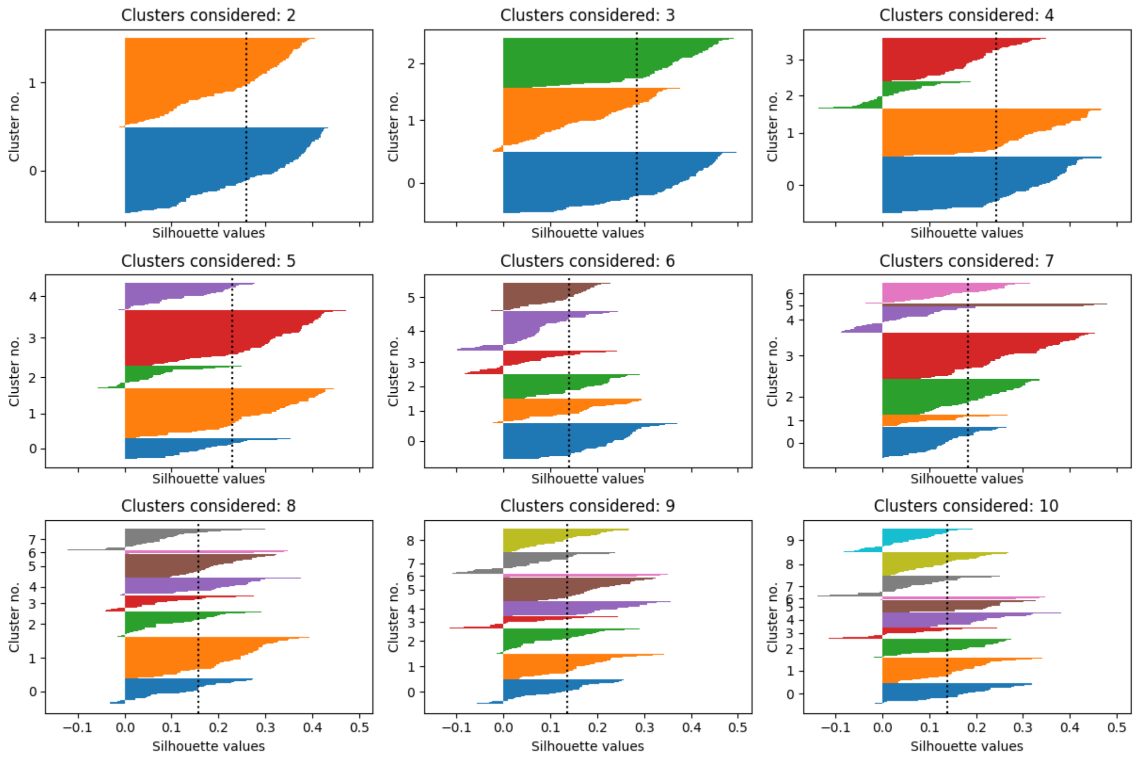

Figure 8: *Silhouette values for the different clusterings considered.*

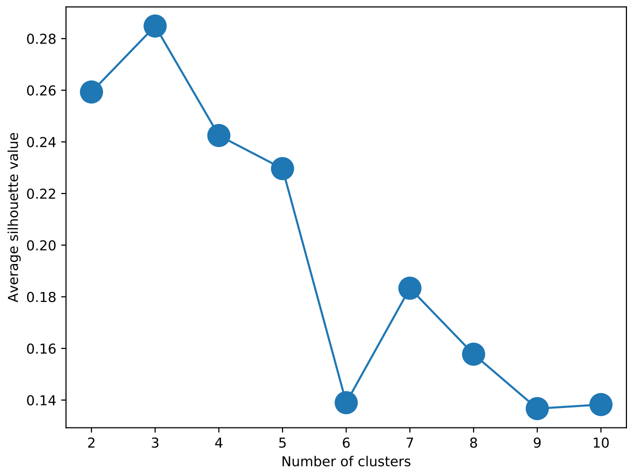

Figure 9: *Average silhouette values for the different clusterings.*

regions found, and the rule reparating them, is given in Fig. 16. We see that the rule we find here, is similar to the simple rule we found in exercise 4.

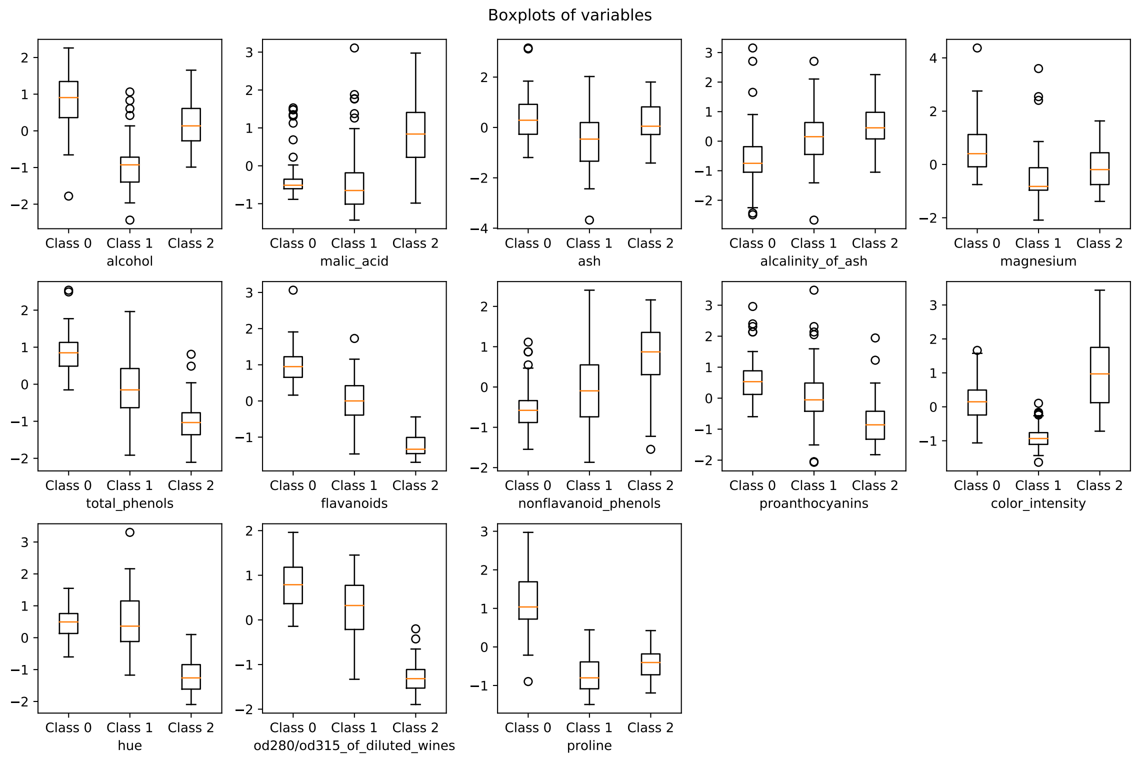

Figure 10: *Box plots for the clusters discovered by KMeans for the wine data set.*

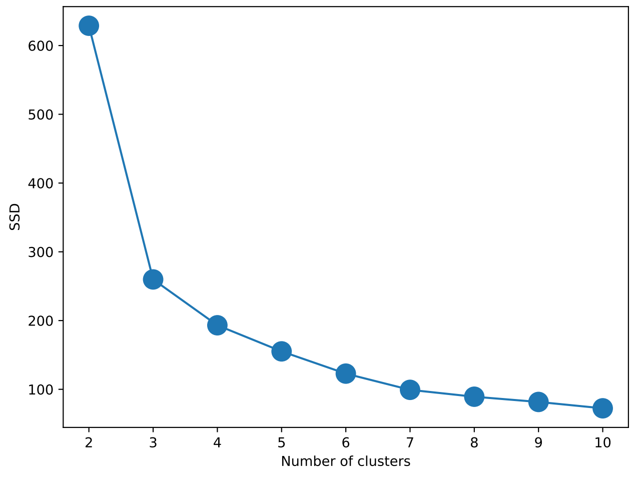

Figure 11: *Sum of squared distances (SSD) of samples to their closest cluster center, when considering the scores from PC1 and PC2.*

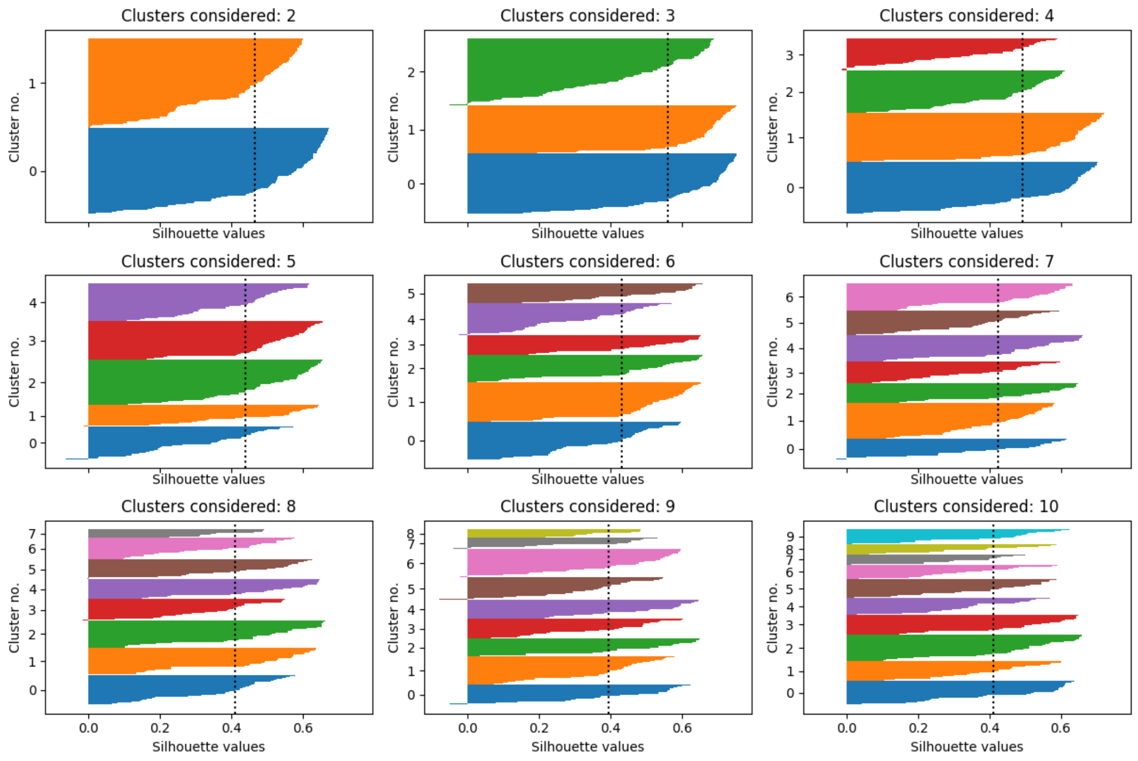

Figure 12: *Silhouette values for the clusterings when considering the scores from PC1 and PC2.*

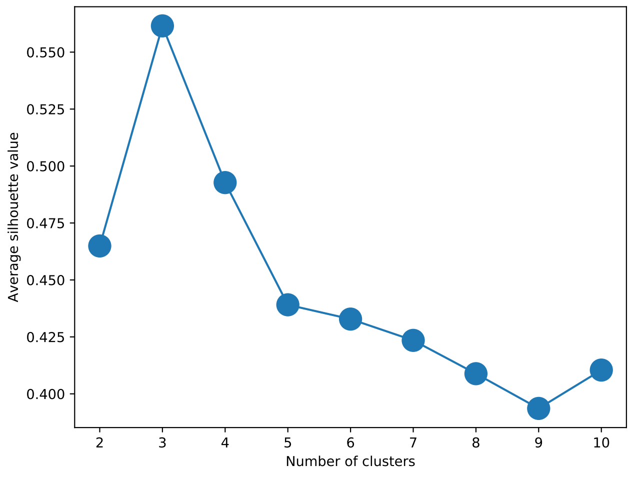

Figure 13: *Average silhouette values for the different clusterings, when considering the scores from PC1 and PC2.*

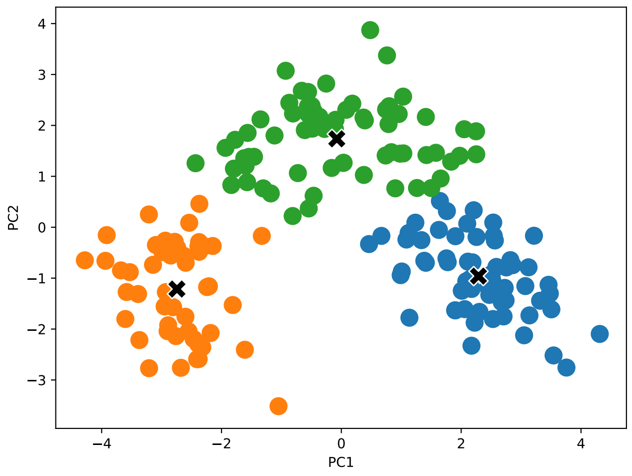

Figure 14: *The scores (circles) and cluster centroids (crosses). The scores have been colored according to the cluster they are assigned to.*

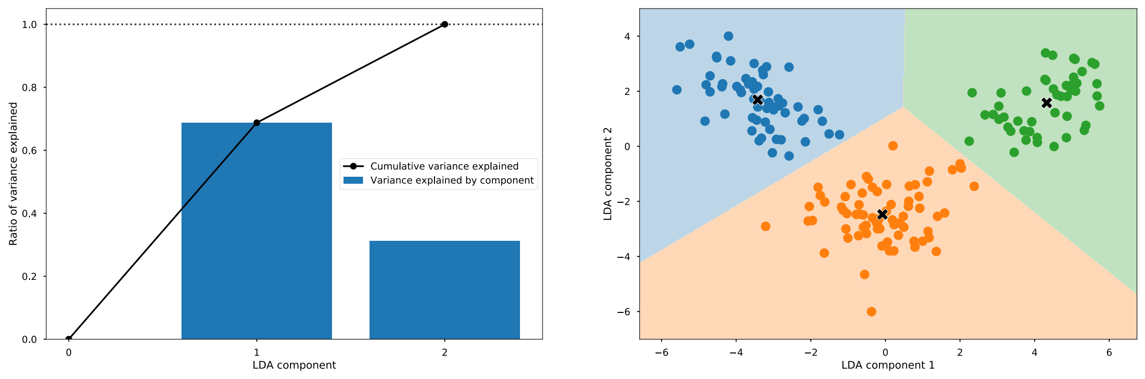

Figure 15: *(Left) The explained variance as a function of the number of LDA components. (Right) The classes found by LDA and the regions defining the classes.*

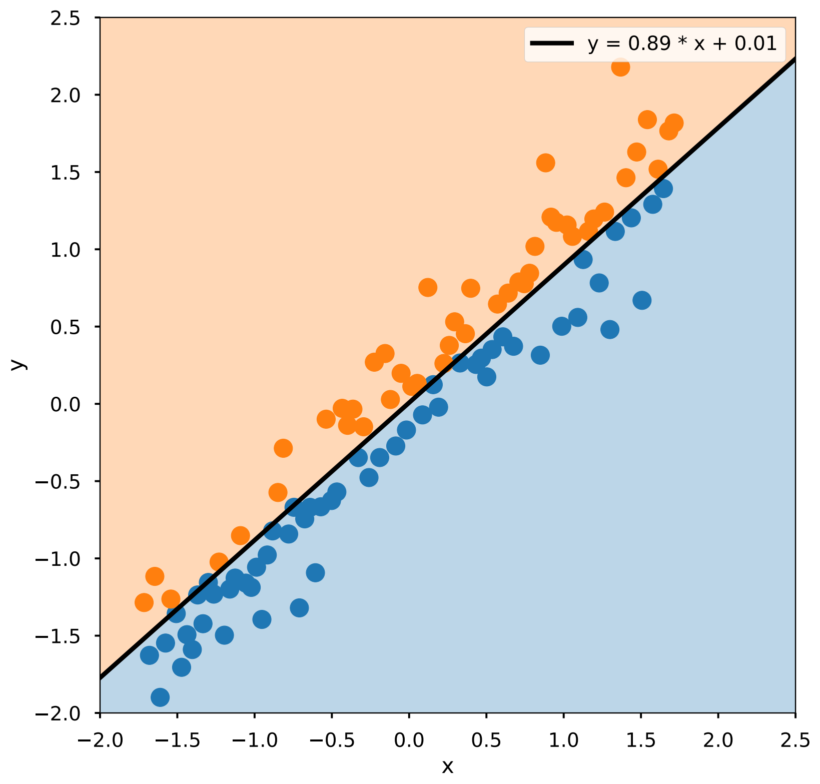

Figure 16: *Results of running LDA for the data set from exercise 4. The solid black line defines the separation of the two classes.*

Exercise – PCA

```python
"""PCA of wine data."""
import numpy as np
import pandas as pd
from sklearn.datasets import load_wine
from sklearn.preprocessing import scale
from sklearn.decomposition import PCA
from matplotlib import pyplot as plt
import matplotlib as mpl
from matplotlib.cm import tab20


COLORS = {
    0: '#1b9e77',
    1: '#d95f02',
    2: '#7570b3',
}


MARKERS = [
    'o', 'v', '^', '<', '>', 's', 'P', 'X', 'D', '*', 'p', '8', 'H'
]


def explore_data(data, class_data):
    """Explore raw data by box plots and histograms."""
    # Let us plot some histograms:
    fig1, axes_grid = plt.subplots(figsize=(12, 8), nrows=3, ncols=5,
                                   constrained_layout=True)
    fig1.suptitle('Histograms of variables')
    axes_flat = axes_grid.flatten()
    for i, variable in enumerate(data.columns):
        for class_id in set(class_data):
            idx = np.where(class_data == class_id)[0]
            data_i = data[variable].iloc[idx]
            axes_flat[i].hist(
                data_i,
                density=True,
                color=COLORS[class_id],
                alpha=0.8,
                edgecolor='black'
            )
        axes_flat[i].set_xlabel(variable)
    for idx, color in COLORS.items():
        axes_flat[-2].hist([], color=color, label='Class {}'.format(idx))
    axes_flat[-1].axis('off')
    axes_flat[-2].axis('off')
    axes_flat[-2].legend(loc='upper center', fontsize='large')
    fig1.savefig('histograms.pdf')
    # Let us check out the averages:
    fig2, axes_grid2 = plt.subplots(
        figsize=(12, 8), nrows=3, ncols=5,
        constrained_layout=True
    )
    fig2.suptitle('Boxplots of variables')
    axes_flat2 = axes_grid2.flatten()
    for i, variable in enumerate(data.columns):
        x = []
        for class_id in set(class_data):
            idx = np.where(class_data == class_id)[0]
            data_i = data[variable].iloc[idx]
            x.append(data_i)
        axes_flat2[i].boxplot(
            x, labels=['Class {}'.format(j) for j in range(len(x))]
        )
        axes_flat2[i].set_xlabel(variable)
    axes_flat2[-1].axis('off')
    axes_flat2[-2].axis('off')
    fig2.savefig('boxplots.pdf')


def plot_explained_variance(pca):
    """Plot the explained variance."""
    fig1, ax1 = plt.subplots(constrained_layout=True)
    variance = [0] + list(np.cumsum(pca.explained_variance_ratio_))
    ax1.plot(variance, marker='o', markersize=14)
    ax1.set(xlabel='Number of principal components',
            ylabel='Explained variance')
    return fig1, ax1


def plot_2d_scores(scores, component1, component2, class_data=None):
    """Plot scores for two PC's"""
    fig1, ax1 = plt.subplots(constrained_layout=True)
    if class_data is None:
        ax1.scatter(scores[:, component1], scores[:, component2], s=100)
    else:
        for class_id in set(class_data):
            idx = np.where(class_data == class_id)[0]
            ax1.scatter(scores[idx, component1], scores[idx, component2],
                        s=100, color=COLORS[class_id],
                        label='Class {}'.format(class_id))
        ax1.legend()
    ax1.set(xlabel='PC{}'.format(component1 + 1),
            ylabel='PC{}'.format(component2 + 1))
    return fig1, ax1


def plot_2d_loadings(pca, component1, component2, variables):
    """Plot loadings for two components."""
    load1 = pca.components_[component1, :]
    load2 = pca.components_[component2, :]
    fig1, ax1 = plt.subplots(constrained_layout=True)
    ax1.set_prop_cycle(mpl.cycler(color=tab20.colors))
    ax1.axhline(y=0, ls=':', color='black')
    ax1.axvline(x=0, ls=':', color='black')
    for i, (loadx, loady, vari) in enumerate(zip(load1, load2, variables)):
        scat = ax1.scatter(loadx, loady, s=150,
                           marker=MARKERS[i], label=vari, edgecolor='black')
        ax1.annotate(vari, (loadx*1.1, loady*1.1),
                     color=scat.get_facecolors()[0], weight='bold')
    ax1.set(xlabel='PC{}'.format(component1 + 1),
            ylabel='PC{}'.format(component2 + 1))
    ax1.set_xlim(-1, 1)
    ax1.set_ylim(-1, 1)
    return fig1, ax1


def initial_pca(data):
    """Run the first PCA of the given data."""
    data_scaled = scale(data)
    pca = PCA(random_state=0)
    pca.fit_transform(data_scaled)

    # Plot explained variance
    fig1, ax1 = plot_explained_variance(pca)
    ax1.axhline(y=0.95, ls=':', color='black')
    fig1.savefig('variance1.pdf')


def second_pca(data, class_data, n_components=None):
    """Run the second PCA of the given data."""
    data_scaled = scale(data)
    pca = PCA(n_components=n_components, random_state=0)
    scores = pca.fit_transform(data_scaled)

    # Plot explained variance
    fig1, ax1 = plot_explained_variance(pca)
    ax1.axhline(y=0.95, ls=':', color='black')
    fig1.savefig('variance2.pdf')

    # Plot the scores for PC1 and PC2:
    fig2, _ = plot_2d_scores(scores, 0, 1, class_data=class_data)
    fig2.savefig('scores.pdf')

    # Plot loadings on PC1 and PC2:
    fig3, _ = plot_2d_loadings(pca, 0, 1, data.columns)
    fig3.savefig('loadings.pdf')
```

```python
    # Save scores:
    pc_name = ['PC{}'.format(i + 1) for i in range(pca.n_components_)]
    scores_data = pd.DataFrame(scores, columns=pc_name)
    scores_data.to_csv('scores.csv')


def main():
    """Run PCA."""
    data_set = load_wine()
    data = pd.DataFrame(data_set['data'], columns=data_set['feature_names'])
    class_data = data_set['target']

    explore_data(data, class_data)
    plt.show()

    initial_pca(data)
    plt.show()

    second_pca(data, class_data, n_components=10)
    plt.show()


if __name__ == '__main__':
    main()
```

Listing 1: *Python code for performing PCA of the wine data set.*

Exercise – KMeans

```python
"""Run KMeans clustering for the wine data set."""
import numpy as np
import pandas as pd
import matplotlib as mpl
from matplotlib import pyplot as plt
from matplotlib.cm import tab20
from sklearn.datasets import load_wine
from sklearn.preprocessing import scale
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples


def run_clustering(X, n_clusters):
    """Do a KMeans clustering with the specified number of clusters."""
    clu = KMeans(
        n_clusters=n_clusters,
        init='k-means++',
        random_state=0,
    )
    y = clu.fit_predict(X)
    return y, clu
```

```python
def do_clusters(data, max_clusters=10):
    """Peform clustering fro a range of cluster numbers."""
    results = []
    for i in range(2, max_clusters + 1):
        y, clu = run_clustering(data, i)
        results.append((i, y, clu))
    return results


def plot_ssd(results):
    """Plot sum of squared distances."""
    fig1, ax1 = plt.subplots(constrained_layout=True)
    ssd = np.array([(i, clu.inertia_) for (i, _, clu) in results])
    ax1.plot(ssd[:, 0], ssd[:, 1], marker='o', markersize=14)
    ax1.set(xlabel='Number of clusters', ylabel='SSD')
    return fig1, ax1


def plot_silhouette(X, y, ax1=None):
    """Plot silhouettes for a given clustering."""
    silh = silhouette_samples(X, y, metric='euclidean')
    avg = silh.mean()
    fig1 = None
    if ax1 is None:
        fig1, ax1 = plt.subplots(constrained_layout=True)
    cluster_labels = np.unique(y)
    pos0 = 0
    yticks = []
    ytick_pos = []
    for i in cluster_labels:
        values = sorted(silh[y == i])
        pos = np.arange(len(values)) + pos0
        ytick_pos.append(pos.mean())
        yticks.append(i)
        pos0 = max(pos) + 1
        ax1.barh(pos, values, edgecolor='none', height=1)
    ax1.axvline(x=avg, ls=':', color='black')
    ax1.set_yticks(ytick_pos)
    ax1.set_yticklabels(yticks)
    ax1.set_xlabel('Silhouette values')
    ax1.set_ylabel('Cluster no.')
    return fig1, ax1, avg


def box_plot(data, class_data, variables):
    """Make some box plots of the variables."""
    fig1, axes_grid = plt.subplots(
        figsize=(12, 8), nrows=3, ncols=5,
        constrained_layout=True
    )
    fig1.suptitle('Boxplots of variables')
    axes_flat = axes_grid.flatten()
    for i, variable in enumerate(variables):
        x = []
        for class_id in sorted(np.unique(class_data)):
            idx = np.where(class_data == class_id)[0]
            x.append(data[idx, i])
        axes_flat[i].boxplot(
            x, labels=['Class {}'.format(j) for j in range(len(x))]
        )
        axes_flat[i].set_xlabel(variable)
    axes_flat[-1].axis('off')
    axes_flat[-2].axis('off')
    return fig1, axes_flat


def bar_plot(data, class_data, variables):
    """Make some box plots of the variables."""
    fig1, ax1_grid = plt.subplots(
        nrows=3,
        sharex=True,
        sharey=True,
        figsize=(12, 8),
        constrained_layout=True
    )
    ax1_flat = ax1_grid.flatten()
    for i, axi in enumerate(ax1_flat):
        axi.axhline(y=0, ls='-', color='black')
        axi.set_prop_cycle(mpl.cycler(color=tab20.colors))
        axi.set_ylabel('Cluster {}'.format(i))
    fig1.suptitle('Averages of variables for clusters')
    for i, _ in enumerate(variables):
        for j, class_id in enumerate(sorted(np.unique(class_data))):
            idx = np.where(class_data == class_id)[0]
            ax1_flat[j].bar(i, data[idx, i].mean())

    ax1_flat[-1].set_xticks(range(len(variables)))
    ax1_flat[-1].set_xticklabels(variables, rotation=30,
                                 rotation_mode='anchor', ha='right')
    return fig1, ax1_flat


def main1():
    """Run KMeans for the wine data set."""
    data_set = load_wine()
    data = pd.DataFrame(data_set['data'], columns=data_set['feature_names'])
    data_scaled = scale(data)

    results = do_clusters(data_scaled)
    fig1, _ = plot_ssd(results)
    fig1.savefig('ssd1.pdf')
    plt.show()

    averages = []
    fig2, axes2 = plt.subplots(
        nrows=3, ncols=3, constrained_layout=True, sharex=True,
        figsize=(12, 8)
    )
    axes_flat = axes2.flatten()
    for (i, y, _), axi in zip(results, axes_flat):
        _, _, avg = plot_silhouette(data_scaled, y, ax1=axi)
        averages.append((i, avg))
        axi.set_title('Clusters considered: {}'.format(i))
    fig2.savefig('silhouettes1.png')
    # Plot average silhouette values:
    fig3, ax3 = plt.subplots(constrained_layout=True)
    averages = np.array(averages)
    ax3.plot(averages[:, 0], averages[:, 1], marker='o', markersize=16)
    ax3.set(xlabel='Number of clusters', ylabel='Average silhouette value')
    fig3.savefig('average_silhouette1.pdf')
    # Make a box plot with information about the clusters we found when
    # we used three clusters:
    _, y, _ = results[1]
    fig4, _ = box_plot(data_scaled, y, data.columns)
    fig4.savefig('cluster_box.pdf')
    fig5, _ = bar_plot(data_scaled, y, data.columns)
    fig5.savefig('cluster_avg.pdf')
    plt.show()


def main2():
    """Run KMeans on the PCA scores."""
    data_raw = pd.read_csv('scores.csv')
    data = data_raw[['PC1', 'PC2']]

    results = do_clusters(data)
    fig1, _ = plot_ssd(results)
    fig1.savefig('ssd2.pdf')
    plt.show()

    averages = []
    fig2, axes2 = plt.subplots(
        nrows=3, ncols=3, constrained_layout=True, sharex=True,
        figsize=(12, 8)
    )
    axes_flat = axes2.flatten()
    for (i, y, _), axi in zip(results, axes_flat):
        _, _, avg = plot_silhouette(data, y, ax1=axi)
        averages.append((i, avg))
        axi.set_title('Clusters considered: {}'.format(i))
    fig2.savefig('silhouettes2.png')
    # Plot average silhouette values:
    fig3, ax3 = plt.subplots(constrained_layout=True)
    averages = np.array(averages)
    ax3.plot(averages[:, 0], averages[:, 1], marker='o', markersize=16)
    ax3.set(xlabel='Number of clusters', ylabel='Average silhouette value')
    fig3.savefig('average_silhouette2.pdf')
    plt.show()
    # Plot the clustering for three clusters:
    fig4, ax4 = plt.subplots(constrained_layout=True)
    _, y, clu = results[1]
    for i in np.unique(y):
        cluster = data[y == i]
        ax4.scatter(cluster['PC1'], cluster['PC2'], s=150)
    # Add cluster centers:
    for row in clu.cluster_centers_:
        ax4.scatter(row[0], row[1], s=200, marker='X',
                    color='black', edgecolor='white')
    ax4.set(xlabel='PC1', ylabel='PC2')
    fig4.savefig('clustering.pdf')
    plt.show()


if __name__ == '__main__':
    main1()
    main2()
```

Listing 2: *Python code for performing KMeans clustering of the wine data set.*

Exercise – LDA

```
"""Load the data for exercise 4 and run LDA."""
from matplotlib import pyplot as plt
from matplotlib.colors import ListedColormap
from matplotlib.cm import tab10
from sklearn.preprocessing import scale
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.preprocessing import LabelEncoder
import numpy as np
import pandas as pd
plt.style.use('seaborn-talk')

raw_data = pd.read_csv('Data/data_exercise4.txt', delim_whitespace=True)
variables = ['x', 'y']
X = scale(raw_data[variables])
# Transform the class information to numbers:
encoder = LabelEncoder()
encoder.fit(raw_data['class'])
y = encoder.transform(raw_data['class'])
# Run LDA:
```

```python
lda = LinearDiscriminantAnalysis()
X_trans = lda.fit_transform(X, y)
print('Number of classes:', len(lda.classes_))
# Predict classes for our original points:
y_hat = lda.predict(X)

# Plot the transformed X:
fig1, ax1 = plt.subplots(constrained_layout=True)
for i in np.unique(y_hat):
    x = X_trans[y_hat == i]
    ax1.scatter(x, np.zeros_like(x), color=tab10.colors[i], s=150,
                label='Class {}'.format(i))
ax1.set(xlabel='LDA component 1')
# Plot the centers of the clusters:
for center in lda.transform(lda.means_):
    ax1.scatter(center[0], 0, s=250, color='black',
                marker='X', edgecolor='white')
ax1.spines['bottom'].set_position('zero')
ax1.get_yaxis().set_visible(False)
for spine in ('left', 'top', 'right'):
    ax1.spines[spine].set_visible(False)
ax1.legend()

# Show the regions:
# Here, we can find the actual line: y = a*x + b
a = -lda.coef_[0][0] / lda.coef_[0][1]
b = -lda.intercept_[0] / lda.coef_[0][1]
boundary = 'y = {:.2f} * x + {:.2f}'.format(a, b)
print('Boundary:', boundary)
fig2, ax2 = plt.subplots(constrained_layout=True)
ax2.set_aspect('equal')
X1, X2 = np.meshgrid(np.linspace(-2, 2.5, 500), np.linspace(-2, 2.5, 500))
Z = lda.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape)
ax2.contourf(X1, X2, Z, alpha=0.3, cmap=ListedColormap(tab10.colors[:2]))
# Add the original samples:
for i in np.unique(y_hat):
    idx = np.where(y_hat == i)[0]
    ax2.scatter(X[idx, 0], X[idx, 1], color=tab10.colors[i], s=150)
# Let us also plot the boundary line:
xb = np.linspace(-2, 2.5, 2)
yb = a * xb + b
ax2.plot(xb, yb, color='black', lw=3, label=boundary)
ax2.set(xlabel='x', ylabel='y')
ax2.legend()
plt.show()
```

Listing 3: *Python code for performing LDA of the data from exercise 4.*