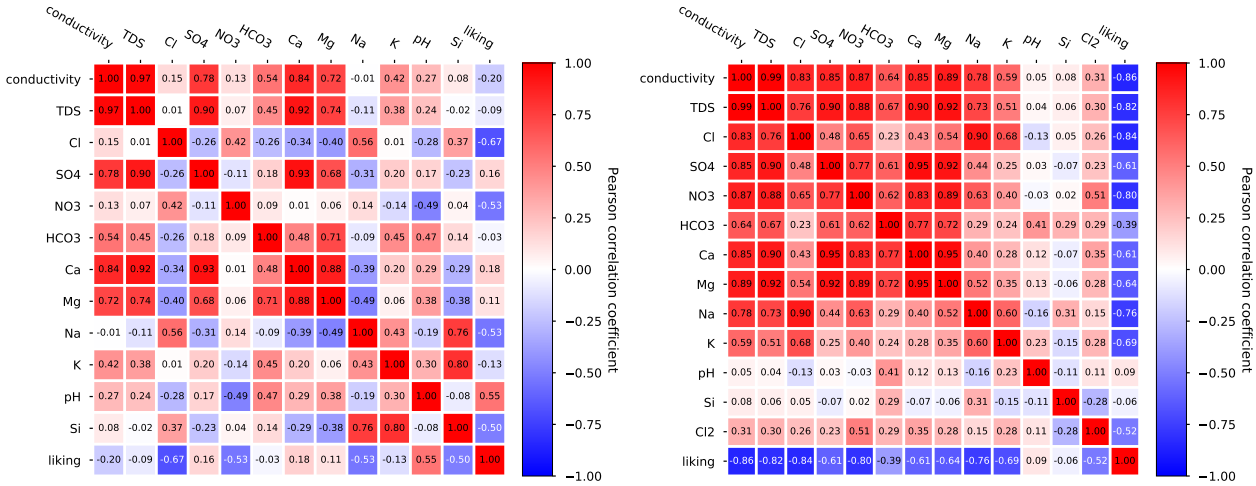
## Exercise 7.1

The Python code for this exercise can be found in listing 1.

(a) Pairwise Pearson correlation coefficients for the variables are shown in Fig. 1a and Fig. 1b for the two data sets. These compare well with the figures given in the original article by Platikanov et al. [1], see their Fig. 1. From these figures we find:




**(a)** *Correlations for data set 1.* **(b)** *Correlations for data set 2.*

**Figure 1:** *Pairwise Pearson correlation coefficients for the variables in the two data sets.*

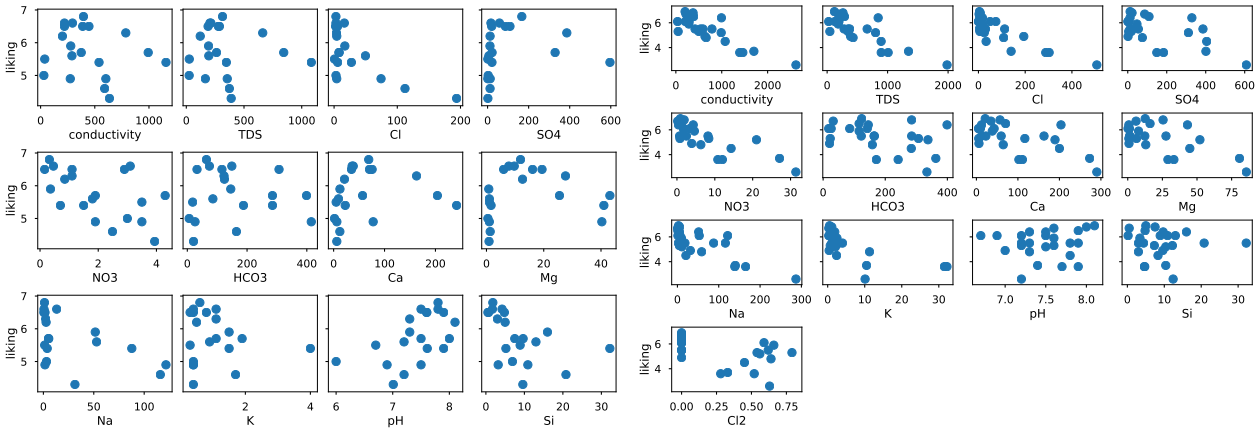
- For data set 1, we find positive correlations between some of the ions (for instance for $Mg^{2+}$, $Ca^{2+}$ and $SO_4^{2-}$) and negative for others (for instance for $Mg^{2+}$ and $Na^{+}$). We also see that there are high correlations between the conductivity and some of the ions, in particular the divalent ions, and between the conductivity and the total dissolved solids (TDS) variable.

  For the mean liking, we find here that it is negatively correlated with the amount of $Cl^{-}$, $NO_3^{-}$, $Na^{+}$, and Si, and positively correlated with the pH.

- For data set 2, we find positive correlations between many of the ions (see the large "red" area in Fig. 1b). We also see that there still are high correlations between the conductivity and the ions, and between the conductivity and TDS.

  For the mean liking, we see strong negative correlations between it and most of the other variables. In this case, there is only a weak positive correlation with the pH.

To further explore the possible correlations between the mean liking and the other variables, we also consider some scatter plots in Fig. 2. For data set 1, we see some correlations between the mean liking and the other variables (for instance for pH), but




**(a)** *Mean liking and variables in data set 1.* **(b)** *Mean liking and variables in data set 2.*

**Figure 2:** *Scatter plots for the mean liking and the other variables in the two data sets.*

the correlations are not very strong, as we already have seen in Fig. 1a. For data set 2, we see stronger correlations between mean liking and the other variables (for instance the amount of $Na^+$), in agreement with what we already have seen in Fig. 1b. In summary:

- In data set 1, the mean liking seems to be negatively correlated with the amount of $Cl^-$, $NO_3^-$, $Na^+$ and Si, and positively correlated with the pH.
- In data set 2, the mean liking seems to be negatively correlated with the amount of the ions, the conductivity, TDS and the amount of $Cl_2$.

(b) We have performed a PCA analysis for the two data sets. The explained variance as a function of principal components can be found in Fig. 3 and Fig. 4, respectively, for the two data sets. We find that:

- For data set 1: The two first principal components explain around 65% of the variance (around 40% for the first principal component and around 25% for the second component).
- For data set 2: The two first principal components explain around 70% of the variance (around 57% for the first principal component and around 13% for the second component).

The scores and loadings for data set 1 are shown in Fig. 5, and in Fig. 6 for data set 2. From these figures, we see the following:

- For data set 1: There is no clear separation into distinct groups in the plot of the scores. The plot of the loadings shows that many of the variables are correlated (for instance the amount of $Na^+$, Si and $Cl^-$). Along principal component 1 (PC1)

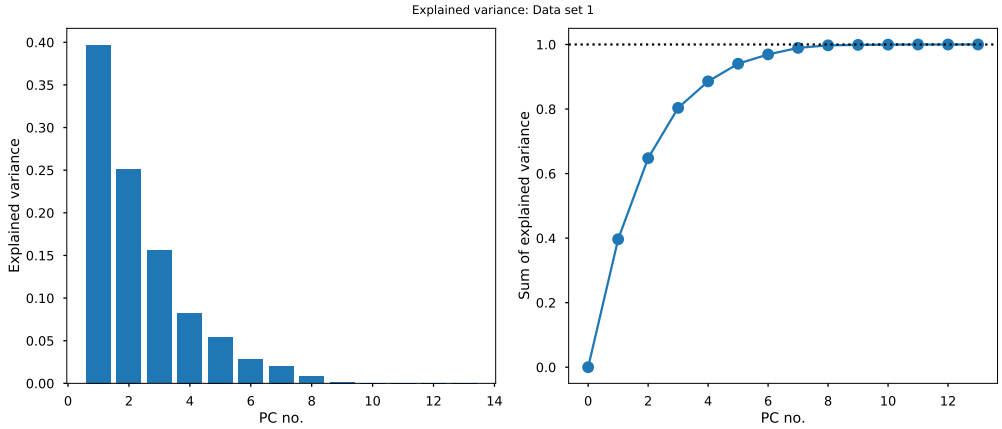



**Figure 3:** *Explained variance as a function of PCA components for data set 1.*

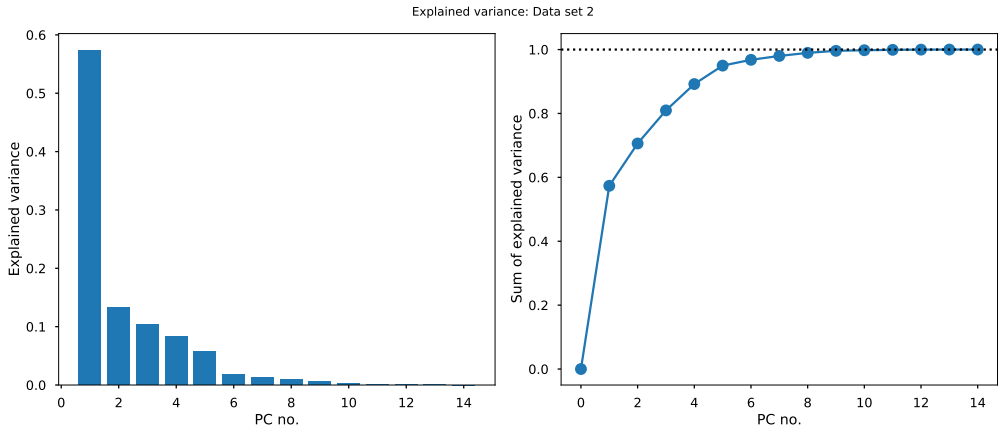



**Figure 4:** *Explained variance as a function of PCA components for data set 2.*

we see that pH and the amount of $K^+$ are positively correlated with the liking. However, $K^+$ is negatively correlated with the liking along principal component 2 (PC2), and pH is less correlated with the liking along this axis.

- For data set 2: Here we see that when we consider PC1 and PC3, there seems to be a separation of the data into the two classes, bottled and tap water samples. Here is also seems that the tab water samples were, on average, less liked than the bottled water samples. Inspection of the loadings show that the mean liking is negatively correlated with the amount of Si, $K^+$, $Na^+$ and $Cl^-$. Further, there are some positive correlations between the mean liking and ions such as $Mg^{2+}$, $Ca^{2+}$, $SO_4^{2-}$ (along PC2).

(c) The predicted and measured mean likings for linear least-squares models are shown in Fig. 7, together with the values of $R^2$, RMSE and RMSECV. The regression coefficients are shown in Fig. 8. We see here that, for both data sets, we get a very high value of $R^2$.

This is maybe as expected since the number of variables we have is comparable to the number of data points. This high $R^2$ does not necessarily mean that our model is good. To investigate this further, we have calculated the average RMSECV value. Since we have a relatively low number of data points, the RMSECV has here been calculated using the leave-one-out approach. We see here that the RMSECV is considerably larger than the RMSE, 2–4 times larger, and this indicates that our model might be over-fitted. We should therefore be careful when using this model for prediction, or, for making general statements.

In any case, based on these models we can try to formulate some interpretations:

- The regression coefficients for the model using data set 1, see Fig. 8a, indicate that the content of $Ca^{2+}$, $SO_4^{2+}$ and $Mg^{2+}$ influences the mean liking most. If we were to create a new brand of bottled water, we could try to focus on having a higher amount of $Ca^{2+}$, while keeping the amounts of $SO_4^{2+}$ and $Mg^{2+}$ low.
- The regression coefficients for the model using data set 2, see Fig. 8b, indicate that the conductivity, TDS and $HCO_3^-$ are most important for the mean liking. If we were to create a new brand of bottled water, we could try to keep the conductivity low, while increasing the TDS and the amount of $HCO_3^-$.

Again, we repeat that our models are probably over-fitted and that we should not put too much faith in the predictions we just made.

(d) The predicted and measured mean likings for PLSR models are shown in Fig. 9, together with the values of $R^2$, RMSE and RMSECV. The regression coefficients are shown in Fig. 10. We see that the $R^2$ values are smaller than the corresponding values from the least-squares fit and that the RMSE values are larger. However, when calculating the RMSECV we see that the values are more consistent for the PLSR models and comparable in size to the RMSE values. We can take this as indicating that our models are *not* over-fitted. In this case, we would prefer the PLSR models over the least-squares models, even though the $R^2$ is smaller for the PLSR models: For the PLSR models, we know that we will make errors when predicting the mean liking for new samples, and we know that these errors will probably be smaller than the errors made with the least-squares models.

The loadings for the two PLS components are shown in Fig. 11. For data set 1, we find that the pH is positively correlated with the mean liking and that ions such as $NO_3^-$ and $Cl^-$ are negatively correlated. For data set 2, we also find a positive correlation of the mean liking with the pH, and negative correlations with ions such as $Na^+$, $Cl^-$, $K^+$, and with the amount of $Cl_2$.

The weights for PLS components 1 and 2 are shown in Fig. 12. We see that these weights are comparable to the weights given by Platikanov et al. [1] (see their Fig. 6), however, we note that our coefficients are normalized differently. Further, we make the following observations:

- The coefficients for PLS component 1 and 2 in data set 1 seems to be the opposite of the coefficients of Platikanov et al. This can be explained with Fig. 11a where we see that our $y$ has negative loadings for both PLS components ($y$ is located at negative positions on both PLS axis in Fig. 11a).
- We see also that our coefficients for PLS component 1 in data set 2 is the opposite of the coefficients of Platikanov et al. Again, this can be explained by noting the sign of the loading for $y$ on PLS component 1 in Fig. 11b, where $y$ is located at a negative position along PLS component axis 1.

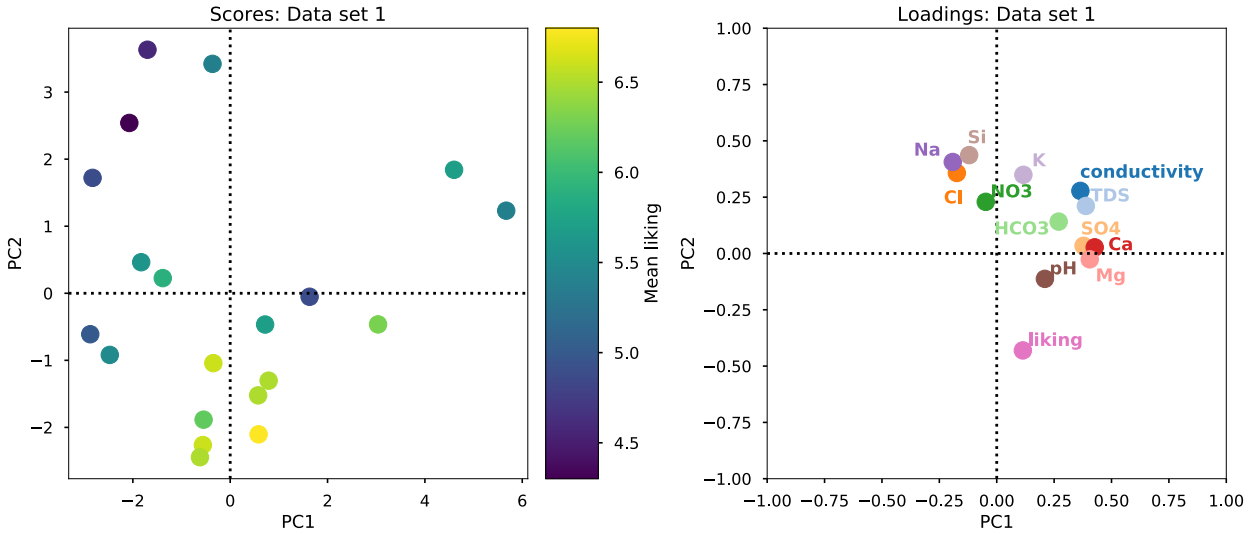
The lesson to be learned from the two points above is that we need both the loadings and the weights when we are to interpret the weights, and if they are positively or negatively correlated with the response ($y$).

Finally, let us try to summarize our findings by predicting how to create a new brand of bottled water:

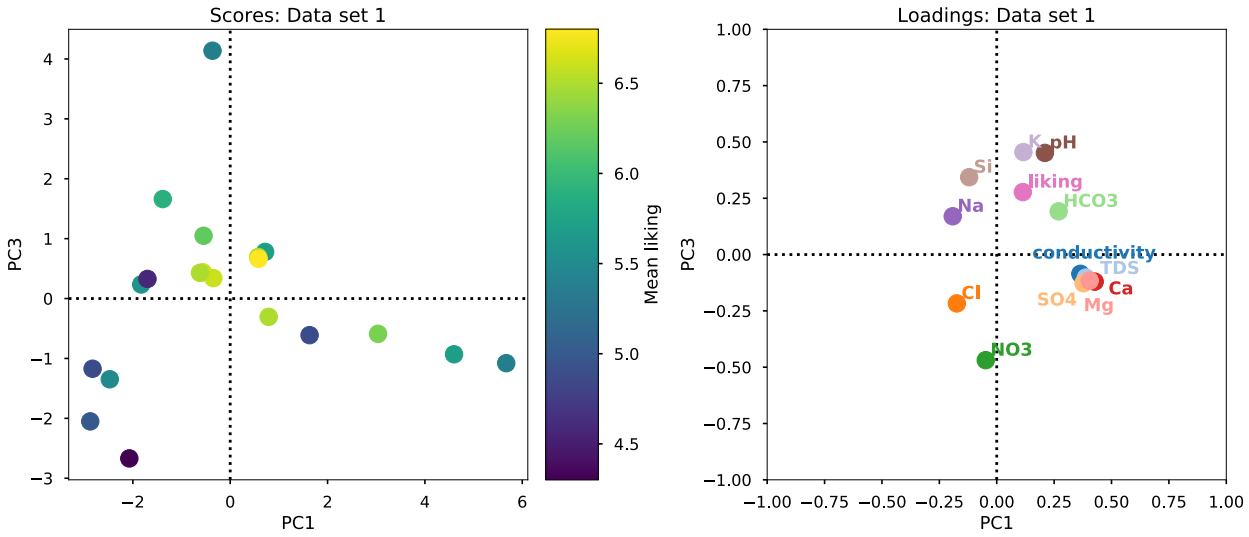
- For data set 1, we see that we should try to keep the pH relatively high, while keeping the amounts of ions such as $Cl^-$, $NO_3^-$ and $Na^+$ low. Further, we should try to limit the amount of Si.
- For data set 2, we see that we actually should try to limit the amount of most of the dissolved ions, and in addition the amount of $Cl_2$. That is, high mineralized waters do not seem to give a high liking. Our finding about the amount of $Cl_2$ should not be a surprise if you have tasted chlorinated water. We also find that the pH is also positively correlated with the mean liking.

# References

[1] Stefan Platikanov et al. "Influence of minerals on the taste of bottled and tap water: A chemometric approach". In: *Water Research* 47.2 (2013), pp. 693–704. DOI: https://doi.org/10.1016/j.watres.2012.10.040.

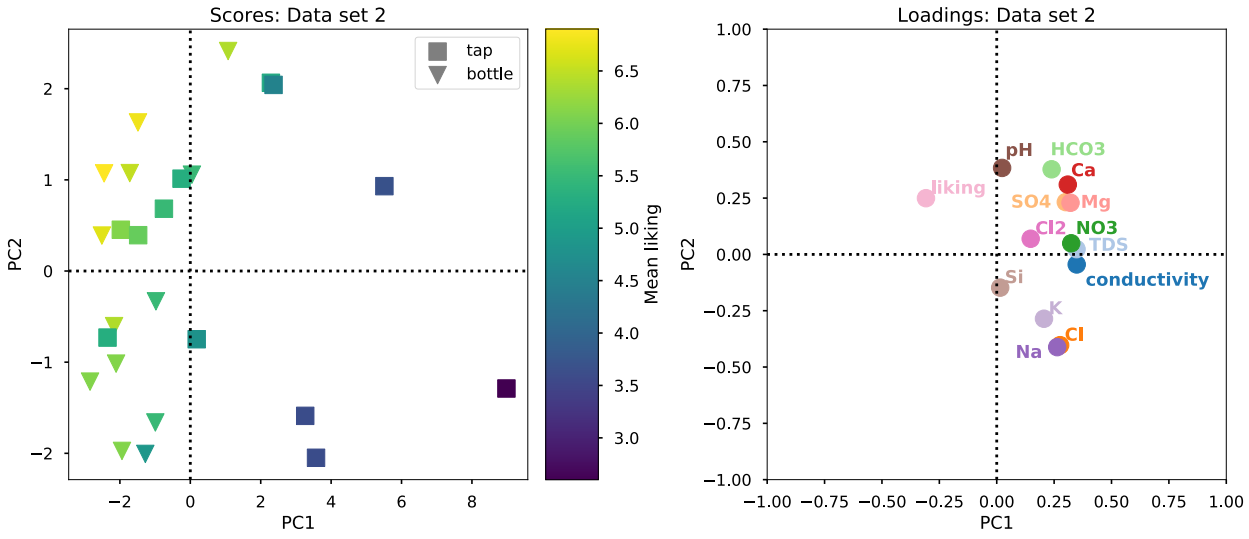



**(a)** *Scores and loadings for PC1 and PC2.*

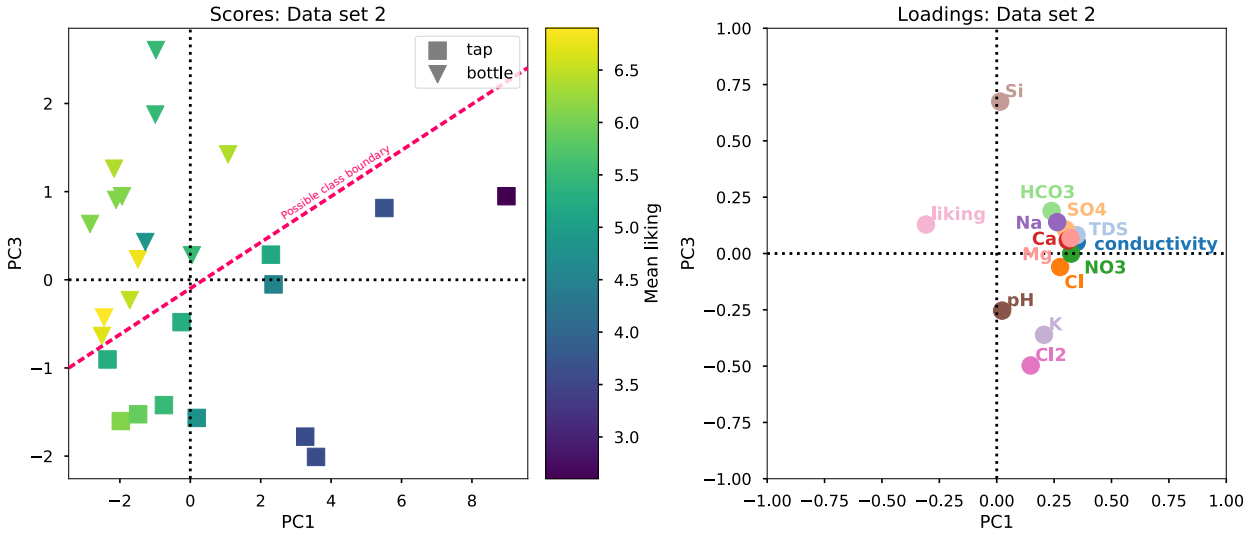



**(b)** *Scores and loadings for PC1 and PC3.*

**Figure 5:** *Scores and loadings for data set 1.*

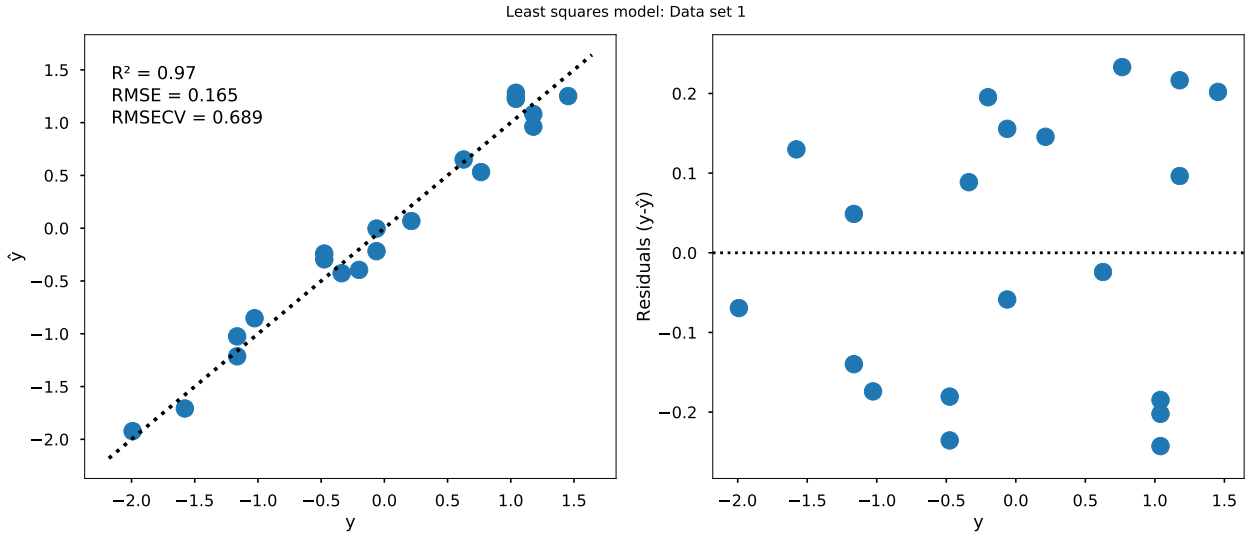



**(a)** *Scores and loadings for PC1 and PC2.*

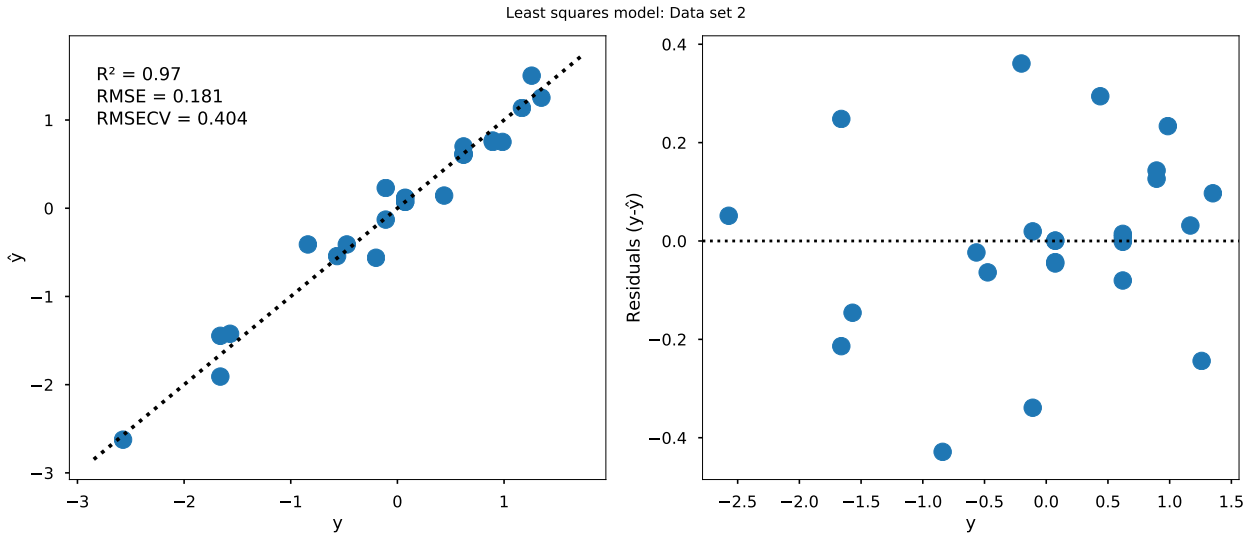



**(b)** *Scores and loadings for PC1 and PC3. A possible separation into bottled samples and tap samples is indicated with the dashed line in the scores plot.*

**Figure 6:** *Scores and loadings for data set 2.*

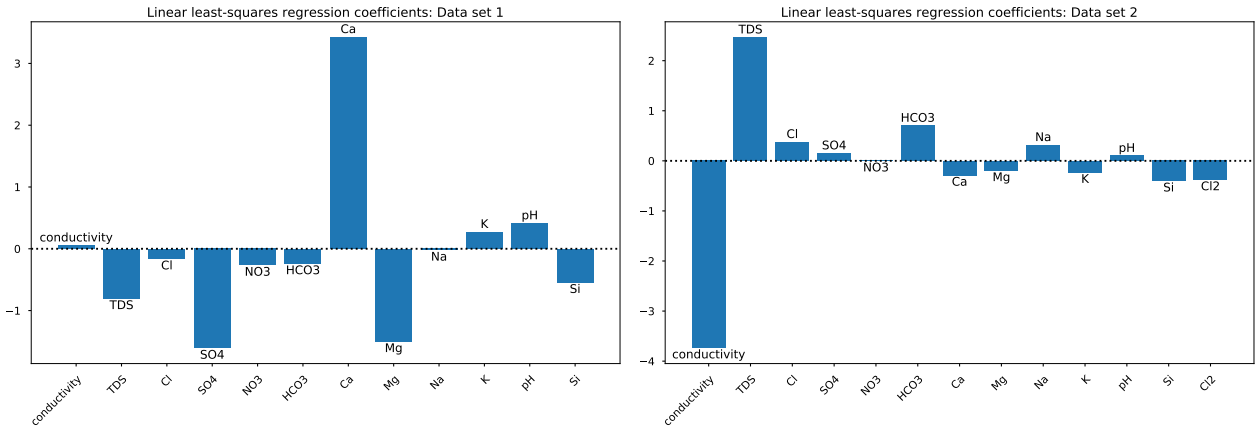



**(a)** *Least-squares model for data set 1.*



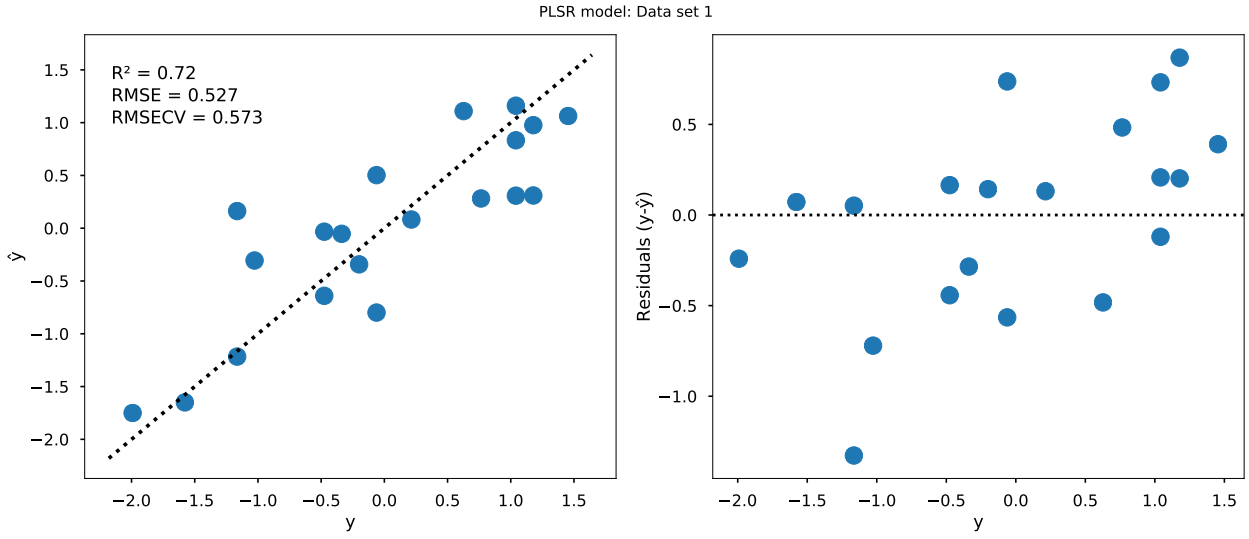

**(b)** *Least-squares model for data set 2.*

**Figure 7:** *Predicted ($\hat{y}$) and measured ($y$) mean likings from linear least-squares models for data set 1 and 2.*

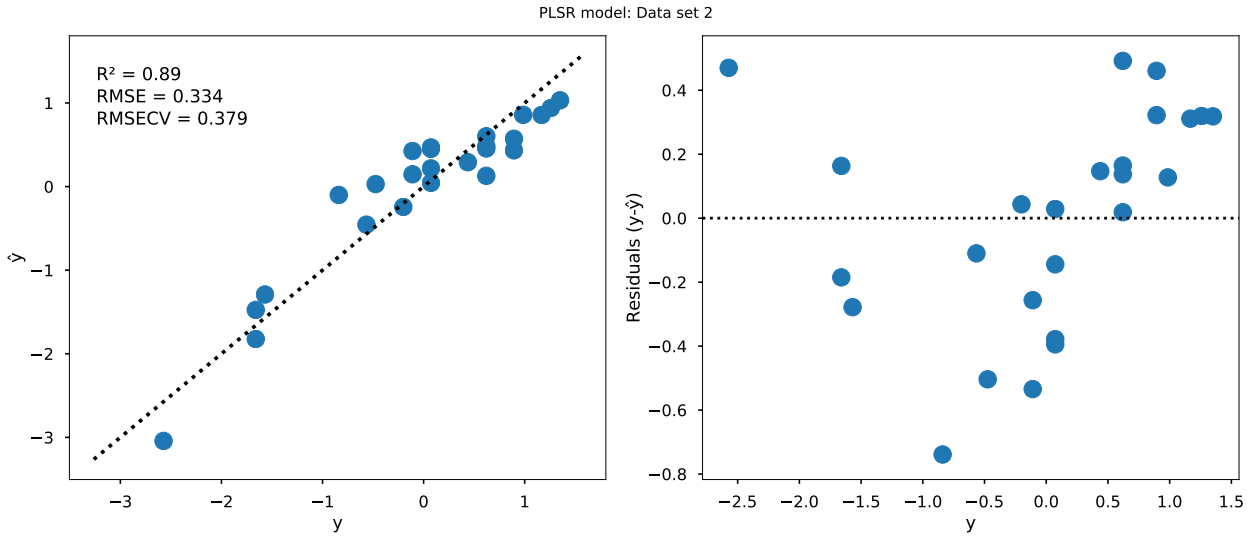



**(a)** *Regression coefficients, data set 1.* **(b)** *Regression coefficients, data set 2.*

**Figure 8:** *Linear least-squares regression coefficients for the linear models fitted to the mean liking of data set 1 and 2.*

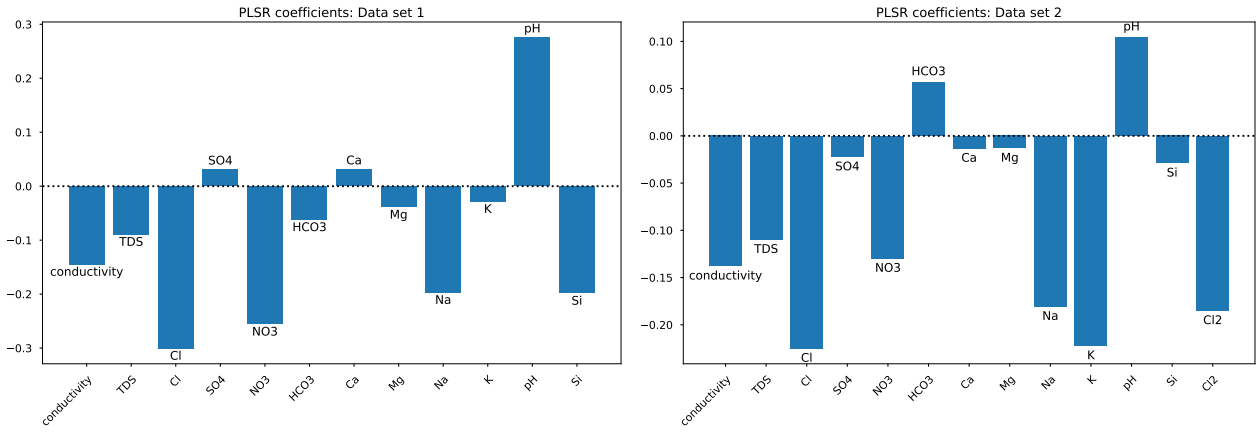



**(a)** *PLSR model for data set 2.*



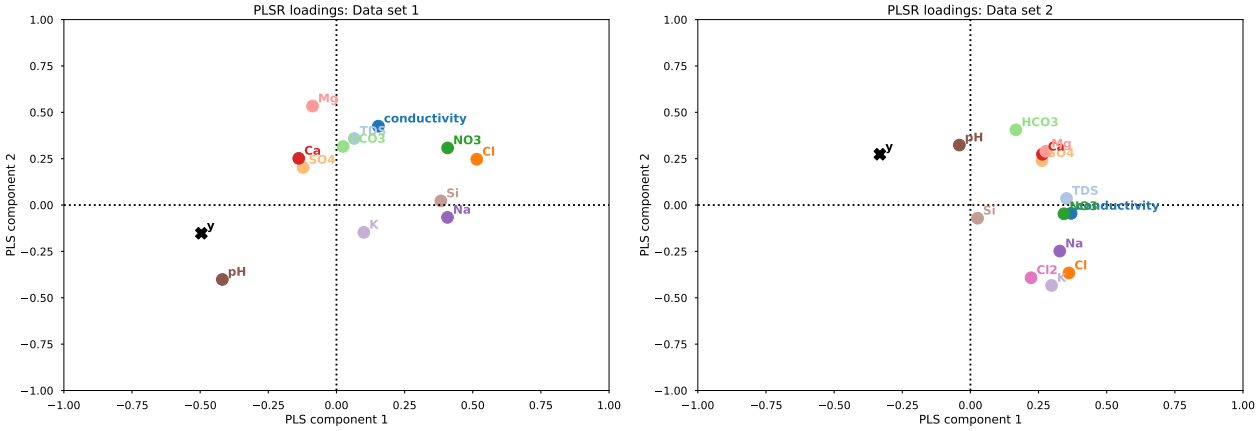

**(b)** *PLSR model for data set 2.*

**Figure 9:** *Predicted ($\hat{y}$) and measured ($y$) mean likings from PLSR models for data set 1 and 2.*



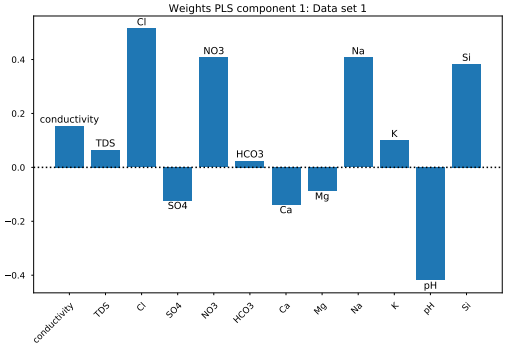

**(a)** *Regression coefficients, data set 1.* **(b)** *Regression coefficients, data set 2.*

**Figure 10:** *PLS regression coefficients for models fitted to the mean liking of data set 1 and 2.*

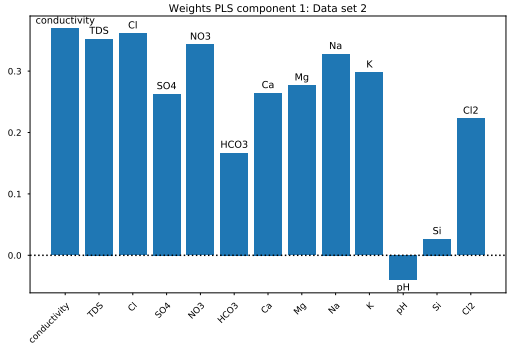



**(a)** *PLS loadings for data set 1.* **(b)** *PLS loadings for data set 2.*

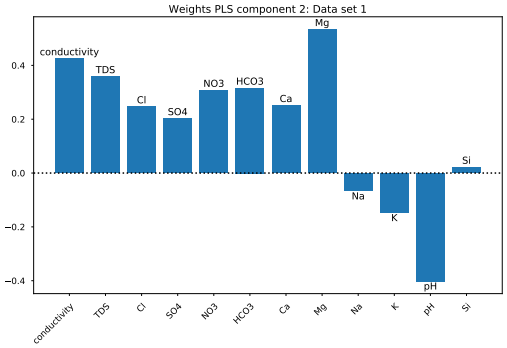
**Figure 11:** *PLS loadings for the PLSR models fitted to the mean liking (y) of data set 1 and 2. Here, we show the weights (the "***R***" matrix) as the loadings for the x-variables.*



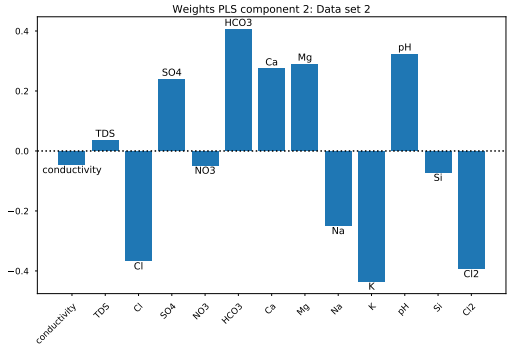

**(a)** *PLS weights for data set 1 (PLS component 1).*




**(b)** *PLS weights for data set 2 (PLS component 1).*




**(c)** *PLS weights for data set 1, (PLS component 2).*




**(d)** *PLS weights for data set 2, (component 2).*

**Figure 12:** *PLS weights for the PLSR models fitted to the mean liking (y) of data set 1 and 2. Here, we show the weights (the "***R***" matrix) for PLS component 1 and PLS component 2.*

### Python solutions

```
"""Solution to exercise 7."""
from matplotlib import pyplot as plt
from matplotlib.ticker import StrMethodFormatter
from matplotlib.cm import tab20
from matplotlib.markers import MarkerStyle
import numpy as np
import pandas as pd
from sklearn.preprocessing import scale
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import PLSRegression
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
plt.style.use('seaborn-talk')


def heatmap(data, row_labels, col_labels,
            cbar_kw=None, cbarlabel='', **kwargs):
    """Create a heat map from a numpy array and two lists of labels.

    Parameters
    ----------
    data : object like :class:`numpy.ndarray`
        A 2D numpy array of shape (N, M).
    row_labels : list of strings
        A list or array of length N with the labels for the rows.
    col_labels : list of strings
        A list or array of length M with the labels for the columns.
    cbar_kw : dict, optional
        A dictionary with arguments to the creation of the color bar.
    cbarlabel : string, optional
        The label for the color bar.
    **kwargs : dict, optional
        Additional arguments for drawing the heat map.

    Returns
    -------
    fig : object like :class:`matplotlib.figure.Figure`
        The figure in which the heatmap is plotted.
    axi : object like :class:`matplotlib.axes.Axes`
        The axis to which the heatmap is added.
    img : object like :class:`matplotlib.image.AxesImage`
        The generated heat map.
    cbar : object like :class:`matplotlib.colorbar.Colorbar`
        The color bar created for the heat map.

    """
    fig, axi = plt.subplots(constrained_layout=True)
```

```
    # Plot the heatmap:
    img = axi.imshow(data, **kwargs)
    # Create colorbars:
    if cbar_kw is None:
        cbar_kw = {}
    cbar = axi.figure.colorbar(img, ax=axi, **cbar_kw)
    cbar.ax.set_ylabel(cbarlabel, rotation=-90, va='bottom')

    # Show ticks using the provided labels:
    axi.set_xticks(np.arange(data.shape[1]))
    axi.set_xticklabels(
        col_labels,
        rotation=-30,
        horizontalalignment='right',
        rotation_mode='anchor'
    )
    axi.set_yticks(np.arange(data.shape[0]))
    axi.set_yticklabels(row_labels)

    # Labels on top:
    axi.tick_params(
        top=True,
        bottom=False,
        labeltop=True,
        labelbottom=False
    )

    # Hide spines off:
    for _, spine in axi.spines.items():
        spine.set_visible(False)

    # Add grid:
    axi.grid(False)
    axi.set_xticks(np.arange(data.shape[1] + 1) - 0.5, minor=True)
    axi.set_yticks(np.arange(data.shape[0] + 1) - 0.5, minor=True)
    axi.grid(which='minor', color='white', linestyle='-', linewidth=3)
    axi.tick_params(which='minor', bottom=False, left=False)
    return fig, axi, img, cbar


def annotate_heatmap(img, data=None, val_fmt='{x:.2f}', textcolors=None,
                     **kwargs):
    """Annotate a heatmap with values.

    Parameters
    ----------
    img : object like :class:`matplotlib.image.AxesImage`
        The heat map image to be labeled.
    data : object like :class:`numpy.ndarray`, optional
        Data used to annotate the heat map. If not given, the
```

```
        data in the heat map image (``img``) is used.
    val_fmt : string, optional
        The format of the annotations inside the heat map.
    textcolors : list of strings, optional
        Colors used for the text. The number of colors provided defines
        a binning for the data values, and values are colored with the
        corresponding color. If no colors are provided, all are colored
        black.
    **kwargs : dict, optional
        Extra arguments used for creating text labels.

    """
    if data is None:
        data = img.get_array()

    # Create arguments for text:
    textkw = kwargs.copy()
    textkw.update(
        {
            'horizontalalignment': 'center',
            'verticalalignment': 'center',
        }
    )

    # Get the formatter:
    formatter = StrMethodFormatter(val_fmt)

    if textcolors is None:
        textcolors = ['black']

    texts = []
    bins = np.linspace(0, 1, len(textcolors) + 1)
    for i in range(data.shape[0]):
        for j in range(data.shape[1]):
            val = img.norm(data[i, j])
            idx = np.digitize(val, bins, right=True)
            idx = max(idx - 1, 0)
            textkw.update(color=textcolors[idx])
            text = img.axes.text(j, i, formatter(data[i, j], None), **textkw)
            texts.append(text)
    return texts


def plot_annotated_heatmap(data, row_labels, col_labels, cbarlabel='',
                           val_fmt='{x:.2f}', textcolors=None,
                           **kwargs):
    """Plot a heat map to investigate correlations.

    Parameters
    ----------
```

```
    data : object like :class:`numpy.ndarray`
        A 2D numpy array of shape (N, M).
    row_labels : list of strings
        A list or array of length N with the labels for the rows.
    col_labels : list of strings
        A list or array of length M with the labels for the columns.
    cbarlabel : string, optional
        The label for the color bar.
    val_fmt : string, optional
        The format of the annotations inside the heat map.
    textcolors : list of strings, optional
        Colors used for the text. The number of colors provided defines
        a binning for the data values, and values are colored with the
        corresponding color. If no colors are provided, all are colored
        black.
    **kwargs : dict, optional
        Arguments used for drawing the heat map.

    Returns
    -------
    fig : object like :class:`matplotlib.figure.Figure`
        The figure in which the heatmap is plotted.
    ax1 : object like :class:`matplotlib.axes.Axes`
        The axis to which the heat map is added.

    """
    fig1, ax1, img, _ = heatmap(
        data,
        row_labels,
        col_labels,
        cbarlabel=cbarlabel,
        **kwargs.get('heatmap', {}),
    )
    annotate_heatmap(
        img,
        val_fmt=val_fmt,
        textcolors=textcolors,
        **kwargs.get('text', {}),
    )
    return fig1, ax1


def find_class(item):
    if 'tap' in item:
        return 'tap'
    return 'bottle'


def load_data():
    """Load the raw data and rename some variables."""
```

```
    # For renaming:
    rename1 = {
        'Labels': 'labels',
        'Conductivity (S/cm)': 'conductivity',
        'TDS (mg/L)': 'TDS',
        'Cl-{}  (mg/L)': 'Cl',
        'SO4{2-}  (mg/L)': 'SO4',
        'NO3{-} (mg/L)': 'NO3',
        'HCO3{-} (mg/L)': 'HCO3',
        'Ca{2+} (mg/L)': 'Ca',
        'Mg{2+} (mg/L)': 'Mg',
        'Na{+} (mg/L)': 'Na',
        'K{+} (mg/L)': 'K',
        'pH': 'pH',
        'Si (mg/L)': 'Si',
        'Mean liking': 'liking',
    }
    rename2 = rename1.copy()
    rename2['Cl2  (mg/L)'] = 'Cl2'
    # Load the raw data:
    data1 = pd.read_csv('Data/table1.csv')
    data2 = pd.read_csv('Data/table2.csv')
    # Rename the variables to save some typing:
    data1 = data1.rename(columns=rename1)
    data2 = data2.rename(columns=rename2)
    # Pick out variables for the two cases:
    xvariables1 = [i for i in data1.columns if i not in {'labels', 'liking'}]
    xvariables2 = [i for i in data2.columns if i not in {'labels', 'liking'}]
    y1 = data1['liking']
    y2 = data2['liking']
    X1 = data1[xvariables1]
    X2 = data2[xvariables2]
    # Add class info for data set 2:
    data2['class'] = data2['labels'].apply(find_class)
    variables = {
        'X1': xvariables1,
        'X2': xvariables2,
        'data1': xvariables1 + ['liking'],
        'data2': xvariables2 + ['liking'],
    }
    return data1, data2, X1, X2, y1, y2, variables


def plot_correlations(data, variables, output=None):
    """Make the plot of correlations."""
    corr = data[variables].corr()
    kwargs = {'heatmap': {'cmap': 'bwr', 'vmin': -1, 'vmax': 1}}
    fig, _ = plot_annotated_heatmap(
        corr,
        variables,
```

```
        variables,
        cbarlabel='Pearson correlation coefficient',
        textcolors=['white', 'black', 'black', 'black'],
        **kwargs
    )
    if output is not None:
        fig.savefig(output, bbox_inches='tight')


def plot_liking(data, xvars, output=None):
    """Plot the liking vs the other variables."""
    nrows = - (-len(xvars) // 4)
    fig, axes = plt.subplots(constrained_layout=True, sharey=True,
                             nrows=nrows, ncols=4)
    axes = axes.flatten()
    for i, axi in enumerate(axes):
        if i % 4 == 0:
            axi.set(ylabel='liking')
        try:
            axi.scatter(data[xvars[i]], data['liking'], s=100)
            axi.set(xlabel=xvars[i])
        except IndexError:
            axi.axis('off')
    if output is not None:
        fig.savefig(output, bbox_inches='tight')


def explore_data(data_sets, variables, xvariables, names):
    """Do explorative plotting."""
    # Make correlation plots:
    for data, vari, namei in zip(data_sets, variables, names):
        plot_correlations(data, vari,
                          output='correlation_{}.pdf'.format(namei))

    # Plot the liking vs. the other variables:
    for data, xvars, namei in zip(data_sets, xvariables, names):
        plot_liking(data, xvars, output='liking_{}.pdf'.format(namei))


def plot_scores_loadings(data, scores, loadings, idx1, idx2,
                         title=None, output=None, class_info=None):
    """Plot PCA scores and loadings."""
    figi, (axi1, axi2) = plt.subplots(constrained_layout=True, ncols=2,
                                      figsize=(14, 6))
    scat = axi1.scatter(scores[:, idx1], scores[:, idx2], marker='o',
                        s=200, c=data['liking'])
    if class_info is not None:
        markers = ['s' if i == 'tap' else 'v' for i in class_info]
        # Update markers:
        # Credit: https://github.com/matplotlib/matplotlib/issues/11155
```

```
        paths = []
        for marker in markers:
            marker_obj = MarkerStyle(marker)
            path = marker_obj.get_path().transformed(
                marker_obj.get_transform()
            )
            paths.append(path)
        scat.set_paths(paths)

        axi1.scatter([], [], marker='s', s=200, label='tap', color='0.5')
        axi1.scatter([], [], marker='v', s=200, label='bottle', color='0.5')
        axi1.legend()
    figi.colorbar(scat, ax=axi1, label='Mean liking')
    axi1.set(xlabel='PC{}'.format(idx1 + 1),
             ylabel='PC{}'.format(idx2 + 1))
    if title is not None:
        axi1.set_title('Scores: {}'.format(title))
    axi1.axhline(y=0, ls=':', color='k')
    axi1.axvline(x=0, ls=':', color='k')

    # Plot loadings for the first two principal components:
    loadings1 = loadings[idx1, :]
    loadings2 = loadings[idx2, :]
    for i, (color, vari) in enumerate(zip(tab20.colors, data.columns)):
        axi2.scatter(loadings1[i], loadings2[i], s=200, color=color)
        axi2.text(
            loadings1[i] + 0.02,
            loadings2[i] + 0.02,
            vari,
            fontsize='x-large',
            fontweight='bold',
            color=color,
        )
    axi2.set(xlabel='PC{}'.format(idx1 + 1),
             ylabel='PC{}'.format(idx2 + 1))
    axi2.set_xlim(-1, 1)
    axi2.set_ylim(-1, 1)
    axi2.axhline(y=0, ls=':', color='k')
    axi2.axvline(x=0, ls=':', color='k')

    if title is not None:
        axi2.set_title('Loadings: {}'.format(title))
    if output is not None:
        figi.savefig(
            'pca_scores_loadings_pc{}_pc{}_{}.pdf'.format(
                idx1 + 1, idx2 + 1, output
            ),
            bbox_inches='tight'
        )
```

```
def run_pca(data_set, title=None, output=None, class_info=None):
    """Run a principal component analysis for the given data set."""
    X = scale(data_set)
    pca = PCA()
    scores = pca.fit_transform(X)

    # Plot the explained variance:
    var = pca.explained_variance_ratio_
    components = range(1, len(var) + 1)
    fig1, (ax11, ax12) = plt.subplots(constrained_layout=True, ncols=2,
                                      figsize=(14, 6))
    ax11.bar(components, var)
    ax11.set(xlabel='PC no.', ylabel='Explained variance')
    ax12.plot([0] + list(components), [0] + list(np.cumsum(var)),
              marker='o', markersize=12)
    ax12.set(xlabel='PC no.', ylabel='Sum of explained variance')
    ax12.axhline(y=1.0, ls=':', color='k')
    ax12.set_ylim(-0.05, 1.05)
    if title is not None:
        fig1.suptitle('Explained variance: {}'.format(title))
    if title is not None:
        print(title)
    else:
        print('')
    print('Variance explained by PC1:', var[0])
    print('Variance explained by PC2:', var[1])
    print('Variance explained by PC3:', var[2])

    # Plot scores for the two first principal components,
    # and for principal components 1 and 3.
    selection = ((0, 1), (0, 2))
    figs = []
    for (idx1, idx2) in selection:
        plot_scores_loadings(data_set, scores, pca.components_, idx1, idx2,
                             title=title, output=output, class_info=class_info
   )
    if output is not None:
        fig1.savefig('pca_explained_variance_{}.pdf'.format(output),
                     bbox_inches='tight')


def add_xy_line(axi, **kwargs):
    """Add a y=x line to the given axes.

    Parameters
    ----------
    axi : object like :class:`matplotlib.axes.Axes`
        The axis to add the y=x line to.
    **kwargs : dict, optional
```

```
        Additional arguments passed to the plotting method.

    Returns
    -------
    line : object like :class:`matplotlib.lines.Line2D`
        The created y=x line.

    """
    lim_min = np.min([axi.get_xlim(), axi.get_ylim()])
    lim_max = np.max([axi.get_xlim(), axi.get_ylim()])
    line, = axi.plot([lim_min, lim_max], [lim_min, lim_max], **kwargs)
    return line


def run_least_squares(xdata, ydata, xvariables, title='', output=None):
    """Run linear least-squares regression."""
    X = scale(xdata)
    y = scale(ydata)
    lsr = LinearRegression()
    lsr.fit(X, y)
    y_hat = lsr.predict(X)
    r2score = r2_score(y, y_hat)
    rmse = np.sqrt(mean_squared_error(y, y_hat))

    # Run cross-validation
    cvscore = cross_val_score(lsr, X, y, scoring='neg_mean_squared_error',
                              cv=len(y))
    cvscore = np.sqrt(-cvscore)
    rmsecv = cvscore.mean()

    fig1, (ax11, ax12) = plt.subplots(constrained_layout=True, ncols=2,
                                      figsize=(14, 6))
    ax11.scatter(y, y_hat, s=200)
    ax12.scatter(y, y - y_hat, s=200)
    ax11.text(0.05, 0.9, 'R² = {:.2f}'.format(r2score),
              transform=ax11.transAxes, fontsize='x-large')
    ax11.text(0.05, 0.85, 'RMSE = {:.3f}'.format(rmse),
              transform=ax11.transAxes, fontsize='x-large')
    ax11.text(0.05, 0.80, 'RMSECV = {:.3f}'.format(rmsecv),
              transform=ax11.transAxes, fontsize='x-large')
    ax11.set(xlabel='y', ylabel='ŷ')
    add_xy_line(ax11, ls=':', color='k', lw=3)
    ax12.set(xlabel='y', ylabel='Residuals (yŷ-)')
    ax12.axhline(y=0, ls=':', color='k')
    fig1.suptitle('Least squares model: {}'.format(title))

    # Plot the coefficients:
    fig2, ax2 = plt.subplots(constrained_layout=True)
    pos = range(len(lsr.coef_))
    ax2.bar(pos, lsr.coef_)
```

```
    ax2.set_xticks(pos)
    ax2.set_xticklabels(
        xvariables,
        rotation=45,
        rotation_mode='anchor',
        horizontalalignment='right'
    )
    for xi, yi, vari in zip(pos, lsr.coef_, xvariables):
        if yi < 0:
            ax2.text(xi, yi - 0.03, vari, ha='center', va='top',
                     fontsize='x-large')
        else:
            ax2.text(xi, yi + 0.03, vari, ha='center', va='bottom',
                     fontsize='x-large')
    ax2.axhline(y=0, ls=':', color='k')
    ax2.set_title(
        'Linear least-squares regression coefficients: {}'.format(title)
    )
    if output is not None:
        fig1.savefig('least_squares_{}.pdf'.format(output),
                     bbox_inches='tight')
        fig2.savefig('least_squares_coefficients_{}.pdf'.format(output),
                     bbox_inches='tight')


def run_plsr(xdata, ydata, xvariables, title='', output=None):
    """Run partial least squares regression."""
    X = scale(xdata)
    y = scale(ydata)

    plsr = PLSRegression(n_components=2)
    plsr.fit(X, y)
    y_hat = plsr.predict(X)
    y_hat = y_hat.reshape(y.shape)
    r2score = r2_score(y, y_hat)
    rmse = np.sqrt(mean_squared_error(y, y_hat))

    # Run cross-validation
    cvscore = cross_val_score(plsr, X, y, scoring='neg_mean_squared_error',
                              cv=len(y))
    cvscore = np.sqrt(-cvscore)
    rmsecv = cvscore.mean()

    fig1, (ax11, ax12) = plt.subplots(constrained_layout=True, ncols=2,
                                      figsize=(14, 6))
    ax11.scatter(y, y_hat, s=200)
    ax12.scatter(y, y - y_hat, s=200)
    ax11.text(0.05, 0.9, 'R² = {:.2f}'.format(r2score),
              transform=ax11.transAxes, fontsize='x-large')
    ax11.text(0.05, 0.85, 'RMSE = {:.3f}'.format(rmse),
```

```
            transform=ax11.transAxes, fontsize='x-large')
ax11.text(0.05, 0.80, 'RMSECV = {:.3f}'.format(rmsecv),
            transform=ax11.transAxes, fontsize='x-large')
ax11.set(xlabel='y', ylabel='ŷ')
add_xy_line(ax11, ls=':', color='k', lw=3)
ax12.set(xlabel='y', ylabel='Residuals (yŷ-)')
ax12.axhline(y=0, ls=':', color='k')
fig1.suptitle('PLSR model: {}'.format(title))

Q = plsr.y_loadings_
B = plsr.coef_
R = plsr.x_rotations_
T = plsr.x_scores_
U = plsr.y_scores_

# Plot weights:
fig2, ax2 = plt.subplots(constrained_layout=True)
fig3, ax3 = plt.subplots(constrained_layout=True)
for i, axi in enumerate((ax2, ax3)):
    weights = R[:, i]
    pos = range(len(weights))
    axi.bar(pos, weights)
    axi.set_xticks(pos)
    axi.set_xticklabels(
        xvariables,
        rotation=45,
        rotation_mode='anchor',
        horizontalalignment='right'
    )
    for xi, yi, vari in zip(pos, weights, xvariables):
        if yi < 0:
            axi.text(xi, yi - 0.005, vari, ha='center', va='top',
                     fontsize='x-large')
        else:
            axi.text(xi, yi + 0.005, vari, ha='center', va='bottom',
                     fontsize='x-large')
    axi.axhline(y=0, ls=':', color='k')
    axi.set_title(
        'Weights PLS component {}: {}'.format(i + 1, title)
    )

# Plot x,y-scores:
fig4, ax4 = plt.subplots(constrained_layout=True)
ax4.scatter(T[:, 0], T[:, 1], marker='o', s=200, label='X-scores')
ax4.scatter(U[:, 0], U[:, 1], marker='X', s=200, label='Y-scores')
ax4.set(xlabel='PLS component 1', ylabel='PLS component 2')
ax4.set_title('PLSR scores: {}'.format(title))
# Plot x-weights loadings:
fig5, ax5 = plt.subplots(constrained_layout=True)
for i, (xi, yi) in enumerate(zip(R[:, 0], R[:, 1])):
```

```
        ax5.scatter(xi, yi, s=200, color=tab20.colors[i])
        ax5.text(
            xi + 0.02,
            yi + 0.02,
            xvariables[i],
            fontsize='x-large',
            fontweight='bold',
            color=tab20.colors[i],
        )
    ax5.set_xlim(-1, 1)
    ax5.set_ylim(-1, 1)
    coeff1 = Q[:, 0]
    coeff2 = Q[:, 1]
    ax5.scatter(coeff1, coeff2, marker='X', s=200, color='k')
    ax5.text(
        coeff1 + 0.02,
        coeff2 + 0.02,
        'y',
        fontsize='x-large',
        fontweight='bold',
        color='k',
    )
    ax5.set(xlabel='PLS component 1', ylabel='PLS component 2')
    ax5.axhline(y=0, ls=':', color='k')
    ax5.axvline(x=0, ls=':', color='k')
    ax5.set_title('PLSR loadings: {}'.format(title))

    fig6, ax6 = plt.subplots(constrained_layout=True)
    coeffs = B.flatten()
    pos = range(len(coeffs))
    ax6.bar(pos, coeffs)
    ax6.set_xticks(pos)
    ax6.set_xticklabels(
        xvariables,
        rotation=45,
        rotation_mode='anchor',
        horizontalalignment='right'
    )
    for xi, yi, vari in zip(pos, coeffs, xvariables):
        if yi < 0:
            ax6.text(xi, yi - 0.005, vari, ha='center', va='top',
                     fontsize='x-large')
        else:
            ax6.text(xi, yi + 0.005, vari, ha='center', va='bottom',
                     fontsize='x-large')
    ax6.axhline(y=0, ls=':', color='k')
    ax6.set_title(
        'PLSR coefficients: {}'.format(title)
    )
```

```
    if output is not None:
        fig1.savefig('plsr_{}.pdf'.format(output),
                     bbox_inches='tight')
        fig2.savefig('plsr_weights_1_{}.pdf'.format(output),
                     bbox_inches='tight')
        fig3.savefig('plsr_weights_2_{}.pdf'.format(output),
                     bbox_inches='tight')
        fig4.savefig('plsr_scores_{}.pdf'.format(output),
                     bbox_inches='tight')
        fig5.savefig('plsr_loadings_{}.pdf'.format(output),
                     bbox_inches='tight')
        fig6.savefig('plsr_coefficients_{}.pdf'.format(output),
                     bbox_inches='tight')


def main():
    """Run exercise 7."""
    data1, data2, X1, X2, y1, y2, variables = load_data()

    data_sets = (data1, data2)
    variable_sets = (variables['data1'], variables['data2'])
    xvariables = (variables['X1'], variables['X2'])
    names = ('data_set_1', 'data_set_2')

    # Do some explorative plotting:
    explore_data(data_sets, variable_sets, xvariables, names)

    # Do PCA on the two data sets:
    run_pca(data1[variables['data1']], title='Data set 1', output='data_set_1'
    )
    run_pca(data2[variables['data2']], title='Data set 2', output='data_set_2'
    ,
            class_info=data2['class'])

    run_least_squares(X1, y1, variables['X1'], title='Data set 1',
                      output='data_set_1')
    run_least_squares(X2, y2, variables['X2'], title='Data set 2',
                      output='data_set_2')

    # Do PLSR for the two data sets:
    run_plsr(X1, y1, variables['X1'], title='Data set 1', output='data_set_1')
    run_plsr(X2, y2, variables['X2'], title='Data set 2', output='data_set_2')


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

**Listing 1:** *Python code for exercise 7.*