Introduction to Statistical Modelling

Joris Vankerschaver

2023/02/25

Table of contents

Preface

Most of the figures in this book are generated with R. If you are reading this book online, you can view the source code for each figure by clicking the little triangle next to the word "Code" above the figure. In the PDF version of this book the code is listed inline with the figure.
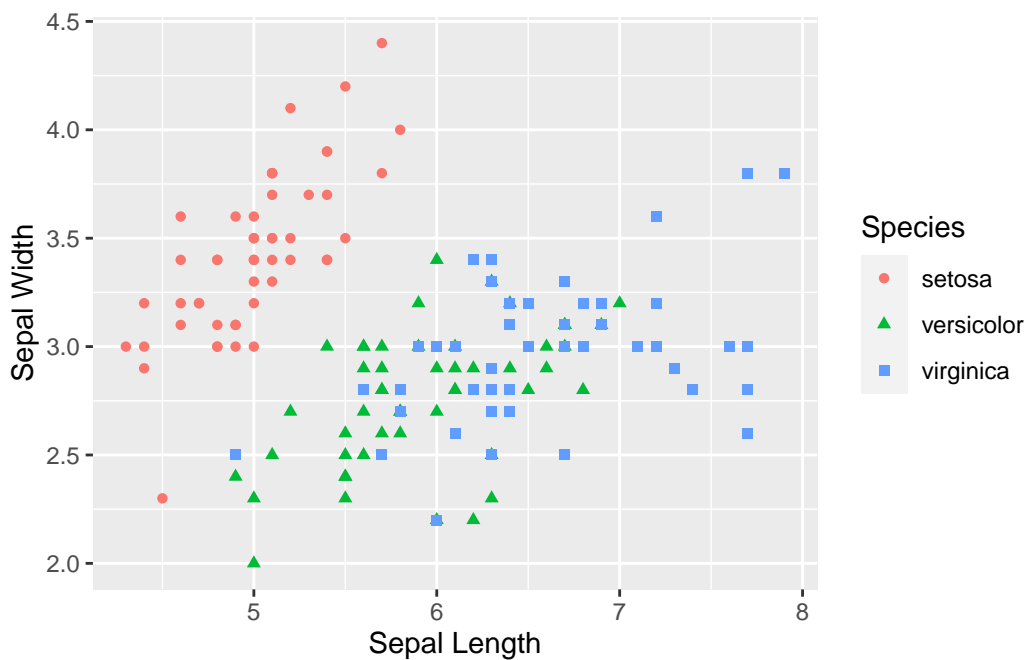

Figure 0.1: Length and width of the sepal petal of 150 Iris plants.

1 Principal component analysis

In the previous chapter we built a linear model to predict the amount of body fat, given measurements of the thickness of the triceps skin fold, the thigh circumference, and the midarm circumference. We saw that the dataset used for this model suffers from multicollinearity, meaning that some of the predictors (or linear combinations of predictors) are correlated with one another. Intuitively speaking, multicollinearity means that some variables don't contribute much to the expressivity of the data: we could omit them and end up with a dataset that is almost as informative as the original one.

To find out which combinations of variables contribute most to the variability of our data, we will turn to *principal component analysis*, one of the mainstays of statistical data analysis. Principal component analysis will allow us to identify the main sources of variability in our dataset, and will tell us which combinations of variables can be omitted with only little impact on the data itself. This allows us to reduce the number of features in the dataset, and makes principal component analysis into what is called a technique for *dimensionality reduction*. This is useful for a number of reasons:

- As a *pre-processing technique*: Many statistical techniques, such as multiple linear regression, do not perform well in the presence of highly correlated features. They either fail to converge outright, or they give unreliable results (for example, model coefficients and predictions that change drastically when the data is slightly perturbed). This is even more of an issue when there are more predictors than data points, a situation that often occurs when analysing gene expression data or spectroscopy data.
- To save on *computational processing time*: analyzing superfluous variables comes with a cost, which can often increase drastically with the number of features. We will see an example of this phenomenon in Section 2.2, where the data points are vectors with 4096 components. After principal component reduction, the dimensionality of the dataset can be reduced to 50-100 components, a reduction by more than 98%.
- To *visualize* the data: for datasets with a limited number of features, we can use a scatter matrix to view the distribution of the features and their relations with one another. Scatter matrices become uninformative, however, as soon as there are more than 4 or 5 features. Moreover, scatter matrices may hide correlations that occur between different linear combinations of variables, as we have seen in the chapter on linear modeling.

1.1 Intuition

Principal component analysis (PCA) finds a low-dimensional approximation to the dataset which retains as much as possible the variability of the original dataset. To understand what is meant by this, let's revisit the body fat dataset of chapter 1. In this dataset, the features are the measurements of the thickness of the triceps skin fold, the thigh circumference, and the midarm circumference. The total body fat is the outcome, but we will not consider it for the time being.
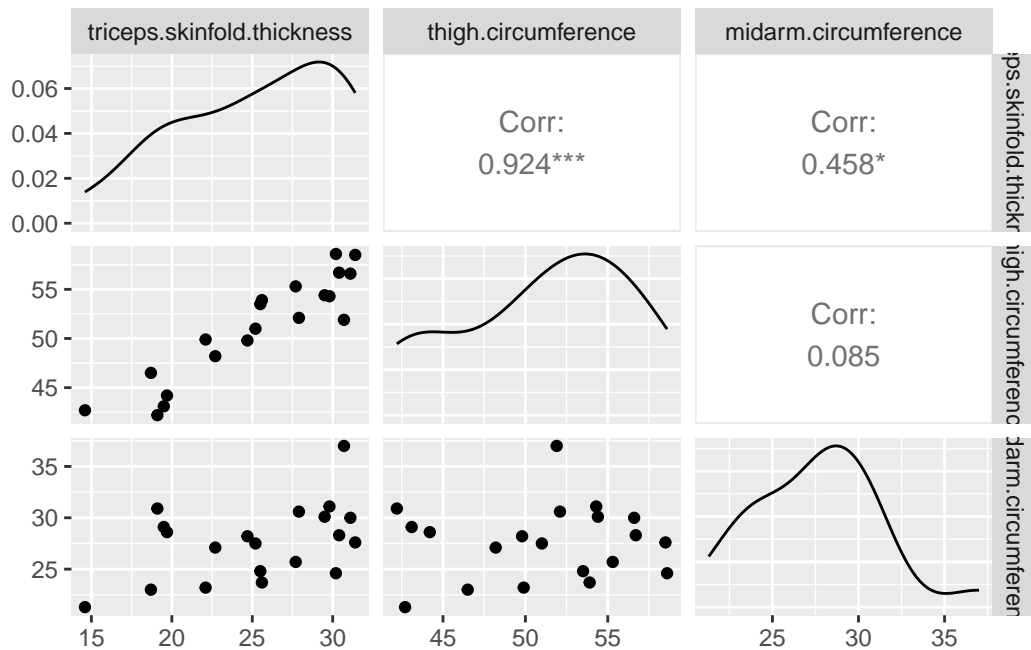

Figure 1.1: Correlations between the different variables in the body fat dataset. The variable `triceps.skinfold.thickness` is strongly correlated with the other two variables in the dataset and hence carries less information.

From the scatter matrix, we see that `triceps.skinfold.thickness` and `thigh.circumference` are highly correlated: if you know one, you can predict the other one reasonably well. This makes it feel like a waste to analyze both: since they carry the same information, we can just as well throw one or the other away, or replace both by a linear combination. Let's do the latter, and introduce a new variable `z1` which is the sum of both. In terms of this variable and `midarm.circumference`, which we leave unchanged, the dataset has only two features that are mildly correlated, as shown on the scatter plot below.
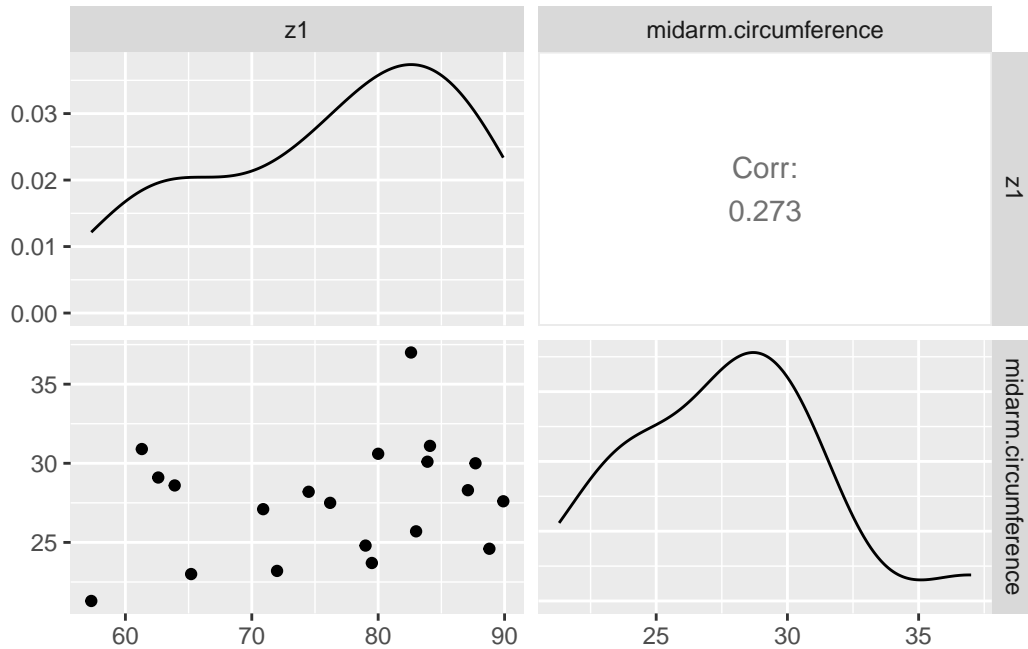
We have succeeded in our aim to reduce the number of features in our dataset from 3 to 2, but a number of questions immediately pop up:

1. What is the meaning of the `z1` variable, and can we find it without looking at a scatter plot?
2. How much information do we lose by considering two variables instead of three? Will the conclusions from the reduced dataset still be valid for the full dataset?

It will turn out that the variable `z1`, which we constructed in an ad-hoc way, is remarkably close to the first principal component of the dataset, and we will discover a way to compute all principal components. We will also see that by discarding more or fewer principal components, as much variability of the original dataset can be retained as is needed.

1.2 Derivation of the PCA algorithm

PCA works by making linear combinations of the original variables so that the total amount of variation is maximal. There is another way of computing principal components, by minimizing the reconstruction error, and it can be shown that both approaches give the same principal components (Bishop (2006), section 12). In this course, we will develop the first method further.

We assume that we have N observations $\mathbf{x}_i$, where each $\mathbf{x}_i$ is a column vector in $\mathbb{R}^D$. We assemble these observations into an $N \times D$ *data matrix* $\mathbf{X}$, where each row of $\mathbf{X}$ is an observation

$\mathbf{x}_i$:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \dots \\ \mathbf{x}_N^T \end{bmatrix}.$$

Keep in mind that the columns of $\mathbf{X}$ are the features (also referred to as independent variables or predictors) of the dataset. In the case of the body fat dataset, $\mathbf{X}$ is a 20×3 matrix, since there are 20 observations, with 3 features for each observation. We will refer to the jth column of $\mathbf{X}$ by the notation $\mathbf{X}_j$, where $j = 1, \dots, D$. Note that $\mathbf{X}_j$ is a column vector, with N entries, and there are D such columns. For the body fat dataset, the columns correspond to the following features:

- $\mathbf{X}_1$: `triceps.skinfold.thickness`
- $\mathbf{X}_2$: `thigh.circumference`
- $\mathbf{X}_3$: `midarm.circumference`

1.2.1 The first principal component

The first principal component, $\mathbf{Z}_1$ is a linear combination of the features, so that the amount of variation in $\mathbf{Z}_1$ is maximized. Let's unpack these ideas one at a time. The fact that $\mathbf{Z}_1$ is a linear combination means that it can be written as

$$\mathbf{Z}_1 = v_1 \mathbf{X}_1 + \dots + v_D \mathbf{X}_D = \sum_{j=1}^{D} v_j \mathbf{X}_j.$$

where the v_j are coefficients that we have to determine. These coefficients are sometimes referred to as the principal component **loadings**, since v_j expresses how much of $\mathbf{X}_j$ is added ("loaded") to the principal component.

We can write this expression for $\mathbf{Z}_1$ in a compact way by assembling all the coefficients v_j into a vector $\mathbf{v}$, called the **loadings vector**:

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \dots \\ v_D \end{bmatrix}.$$

With this expression, $\mathbf{Z}_1$ can be written as a matrix-vector product:

$$\mathbf{Z}_1 = \mathbf{X}\mathbf{v}. \tag{1.1}$$

We will often use this way of expressing $\mathbf{Z}_1$ as a matrix-vector product, since it makes subsequent calculations easier.

Before we go on to determine the loadings v_j, let's focus on the geometry behind Equation 1.1. Each component of $\mathbf{Z}_1$ can written as

$$(\mathbf{Z}_1)_i = \mathbf{x}_i^T \mathbf{v}.$$

This is the dot product of the ith observation $\mathbf{x}_i$ with the loadings vector $\mathbf{v}$. This dot product tells us how much of the vector $\mathbf{x}_i$ is parallel to $\mathbf{v}$, as shown in Figure 1.2. For example, a data point that is at right angles to $\mathbf{v}$ will have dot product 0 (no component at all along $\mathbf{v}$), while one that is parallel to $\mathbf{v}$ will have a dot product that is maximal in magnitude.
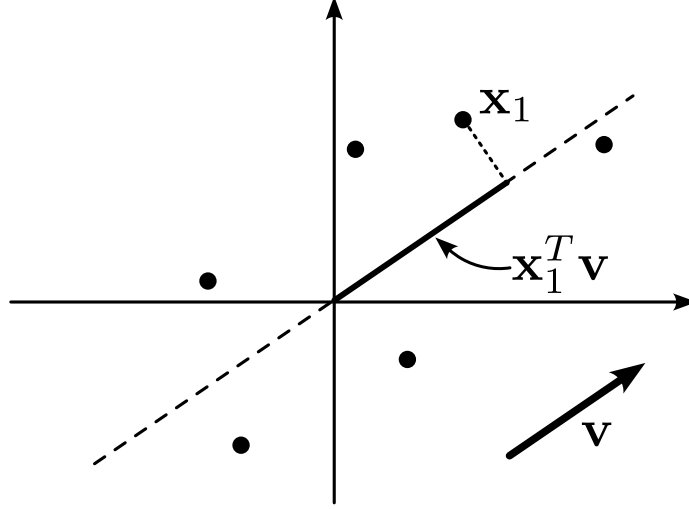

Figure 1.2: The ith component of the first principal component $\mathbf{Z}_1$ is the length of the orthogonal projection of $\mathbf{x}_i$ onto the line in the direction of $\mathbf{v}$.

In other words, we obtain $\mathbf{Z}_1$ by taking a fixed vector $\mathbf{v}$ and projecting all of our data points on the line through the origin in the direction of $\mathbf{v}$. If we choose another vector, $\mathbf{w}$, we obtain a different projection, as indicated on Figure 1.3.

Our goal is now to find the loadings vector $\mathbf{v}$ so that the variance of the projected dataset is maximal. To make this problem well-posed, we will assume that $\mathbf{v}$ has unit norm:

$$\mathbf{v}^T \mathbf{v} = 1. \tag{1.2}$$

If we did not impose this constraint, we could increase the amount of variance simply by making $\mathbf{v}$ longer.

The mean of the projected data is given by

$$\bar{\mathbf{Z}}_1 = \frac{1}{N} \sum_{j=1}^{N} (\mathbf{Z}_1)_j = \frac{1}{N} \sum_{j=1}^{N} \mathbf{x}_j^T \mathbf{v} = \bar{\mathbf{x}}^T \mathbf{v},$$
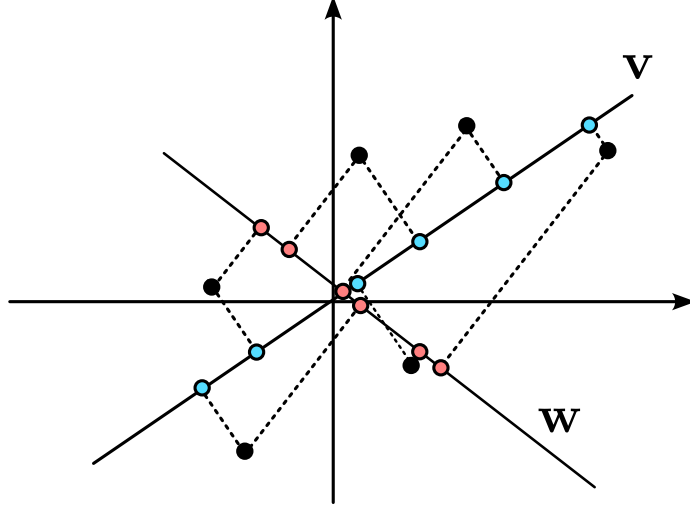
Figure 1.3: Two different projections, onto the loadings vector $\mathbf{v}$ (blue) and $\mathbf{w}$ (red).

where $\bar{\mathbf{x}}$ is the (sample) mean of the original data points. In other words, the mean $\bar{\mathbf{Z}}_1$ is just the mean $\bar{\mathbf{x}}$ of the data points, projected onto $\mathbf{v}$.

The variance of the projected data is given by

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} \left((\mathbf{Z}_1)_i - \bar{\mathbf{Z}}_1 \right)^2 = \frac{1}{N} \sum_{i=1}^{N} \left(\mathbf{x}_i^T \mathbf{v} - \bar{\mathbf{x}}^T \mathbf{v} \right)^2.$$

This expression can be rewritten as a matrix product:

$$\sigma^2 = \mathbf{v}^T \mathbf{S} \mathbf{v}, \tag{1.3}$$

where $\mathbf{S}$ is the covariance matrix, given by

$$\mathbf{S} = \frac{1}{N} \sum_{i=1}^{N} \left(\mathbf{x}_i \mathbf{x}_i^T - \bar{\mathbf{x}} \bar{\mathbf{x}}^T \right). \tag{1.4}$$

We are now ready to translate our problem into a mathematical form, so that we can solve it. To find the first principal component $\mathbf{Z}_1$, we want to find a loadings vector $\mathbf{v}$ so that the projected variance σ^2, given in Equation 1.3, is maximized. In addition, we want $\mathbf{v}$ to have unit length, as in Equation 1.2. In mathematical terms:

$$\mathbf{v} = \operatorname{argmax} \mathbf{v}^T \mathbf{S} \mathbf{v}, \quad \text{such that } \mathbf{v}^T \mathbf{v} = 1.$$

We can solve this optimization problem using the theory of Lagrange multipliers. If we introduce a Lagrange multiplier λ for the unit-length constraint, then the desired vector $\mathbf{v}$ is given by

$$\mathbf{v} = \operatorname{argmax} L(\mathbf{v})$$

where L is given by

$$L(\mathbf{v}) = \mathbf{v}^T\mathbf{S}\mathbf{v} - \lambda(\mathbf{v}^T\mathbf{v} - 1).$$

A necessary condition for $\mathbf{v}$ to be a maximum of L is that the gradient vanishes at $\mathbf{v}$. Taking the gradient of L with respect to $\mathbf{v}$ and setting the resulting expression equal to zero gives

$$\mathbf{S}\mathbf{v} = \lambda\mathbf{v}. \tag{1.5}$$

This is a very important result: it tells us that the $\mathbf{v}$ we are looking for is an **eigenvector** of the matrix $\mathbf{S}$, with corresponding **eigenvalue** λ. This will hold true generally, not just for the first principal component: finding the principal components of a data set will involve solving an eigenvalue problem, and selecting the largest eigenvalues.

Last, we have to find the Lagrange multiplier λ. This can be done by multiplying Equation 1.5 from the left by $\mathbf{v}^T$ to get

$$\mathbf{v}^T\mathbf{S}\mathbf{v} = \lambda\mathbf{v}^T\mathbf{v} = \lambda,$$

where we have used the unit-length constraint Equation 1.2.

We see that the Lagrange multiplier λ is precisely the variance σ^2 of the first principal component $\mathbf{Z}_1$. For this reason, we will refer to the eigenvalue λ as the amount of *retained variance*, since it expresses how much variance is captured by projecting the entire dataset onto the direction $\mathbf{v}$.

To sum up, the first principal component $\mathbf{Z}_1$ is a linear combination of the original features (columns) of our dataset, chosen so that the variance of $\mathbf{Z}_1$ is maximal. We can find $\mathbf{Z}_1$ by looking for the largest eigenvalue λ of the covariance matrix, with unit length eigenvector $\mathbf{v}$, and projecting the data matrix $\mathbf{X}$ onto $\mathbf{v}$.

1.2.2 The remaining principal components

Now that we've computed the first principal component, how do we compute the others? It probably won't come as a surprise that the next principal components, $\mathbf{Z}_2$, $\mathbf{Z}_3$, and so on, involve the amount of variation that is left in the data after $\mathbf{Z}_1$ has been removed, and that they involve the second, third, … largest eigenvalues.

Assuming that we have computed $\mathbf{Z}_1$ as in the previous section, and denote the loadings vector by $\mathbf{v}_1$. Recall that $\mathbf{v}_1$ points in the direction of the largest variance.

To find the next principal component, we consider the variability in the dataset that is not already accounted for by $\mathbf{Z}_1$. More precisely, we look for a loadings vector $\mathbf{v}_2$ which is orthogonal to $\mathbf{v}_1$, has unit length, and maximizes the amount of variability $\mathbf{v}_2^T\mathbf{S}\mathbf{v}_2$. By a similar reasoning as in the previous section, one can show that this $\mathbf{v}_2$ is an eigenvector associated

with the second largest eigenvalue of $\mathbf{S}$. The projection of the dataset onto this loadings vector then gives us the second principal component:

$$\mathbf{Z}_2 = \mathbf{X}\mathbf{v}_2.$$

This procedure can be applied to find all D principal components and results in the following algorithm to compute the principal components:

1. Compute the sample covariance matrix $\mathbf{S}$ using Equation 1.4.
2. Compute the eigenvalues of $\mathbf{S}$ and order them from largest to smallest: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_D$.
3. Find the corresponding eigenvectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_D$ and normalize them to unit length, if necessary. These vectors are the loading vectors and they point in the directions of highest variance.
4. Project the dataset onto the loading vectors to obtain the principal components $\mathbf{Z}_1, \mathbf{Z}_2, \ldots, \mathbf{Z}_D$.

Typically, we do not have to compute the principal components by hand: most data analysis packages will do this for us, either via a builtin command, such as in R (prcomp) or minitab, or via an extra package, such as scikit-learn (Python) or xlstat (Excel). It is instructive, however, to know the principles behind PCA, so that you can interpret and understand the results.

> **ⓘ Note**
>
> Software packages that compute the principal components of a dataset typically do not compute the eigenvalues and eigenvectors of the covariance matrix, despite our derivation above. Instead, they rely on the so-called *singular value decomposition* (SVD) of the data matrix (after centering). The SVD is typically more accurate and easier to compute, and the principal components obtained in this way agree with the ones computed using the eigendecomposition (to within numerical roundoff).

1.2.3 Worked-out example

For this example, we will calculate the principal components in two different ways. We will first compute the principal components by hand, by solving an eigenvalue problem. This is possible because the data are two-dimensional, and solving the characteristic equation for a two- or three-dimensional matrix can be done by hand. This is not practical for real-world datasets, which often contains dozens, thousands, or millions of features, and we will therefore also cover computing the principal components using R.

Our dataset consists of 100 observations, where each observation has two components. The dataset is shown below and has been carefully constructed so that the covariance matrix $\mathbf{S}$

and mean $\bar{\mathbf{x}}$ are exactly equal to

$$\mathbf{S} = \begin{bmatrix} 5 & 2 \\ 2 & 2 \end{bmatrix}, \quad \text{and} \quad \bar{\mathbf{x}} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \tag{1.6}$$
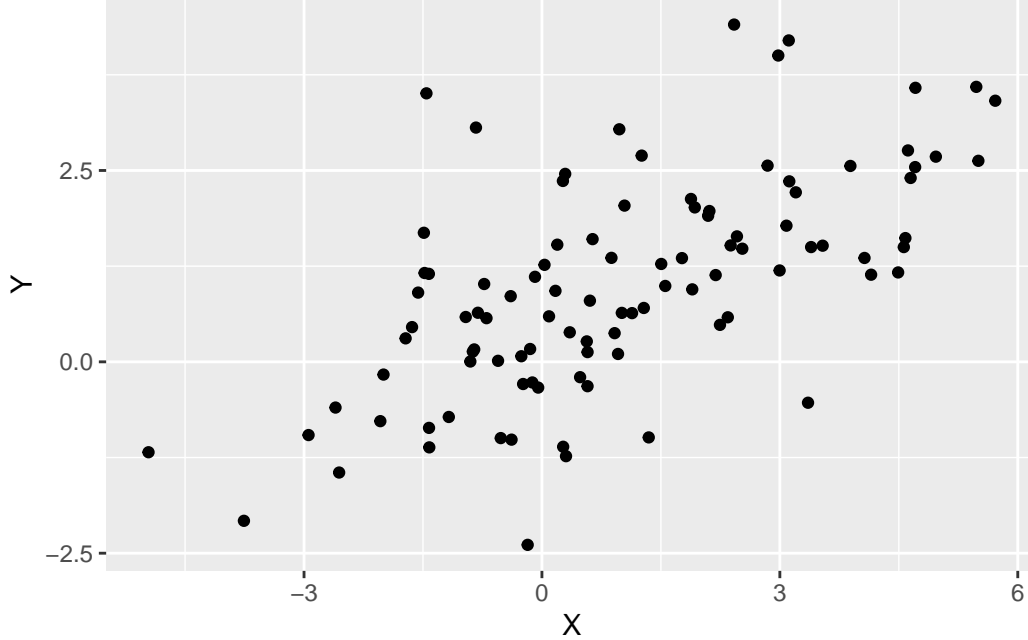
To find the loading vectors, we must find the eigenvalues of $\mathbf{S}$, which we can do via the characteristic equation:

$$\det(\mathbf{S} - \lambda \mathbf{I}) = 0.$$

Substituting the expression given in Equation 1.6 for the covariance matrix and expanding the determinant gives

$$\det \begin{bmatrix} 5 - \lambda & 2 \\ 2 & 2 - \lambda \end{bmatrix} = (5 - \lambda)(2 - \lambda) - 4 = 0.$$

The roots of this equation are $\lambda_1 = 6$ and $\lambda_2 = 1$. The corresponding eigenvectors are

$$\mathbf{v}_1 = \frac{1}{\sqrt{5}} \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \quad \mathbf{v}_2 = \frac{1}{\sqrt{5}} \begin{bmatrix} -1 \\ 2 \end{bmatrix}.$$

These are our loading vectors, and they indicate the direction in which the data varies the most (for $\mathbf{v}_1$) and the "second-most" (for $\mathbf{v}_2$). Figure Figure 1.4 shows the dataset again, now with the two loading vectors superimposed. Each loading vector has been rescaled by multiplying it with the square root of the corresponding eigenvalue. Why the square root? The eigenvalue itself represents the *variance* in that direction, the square root the standard deviation.
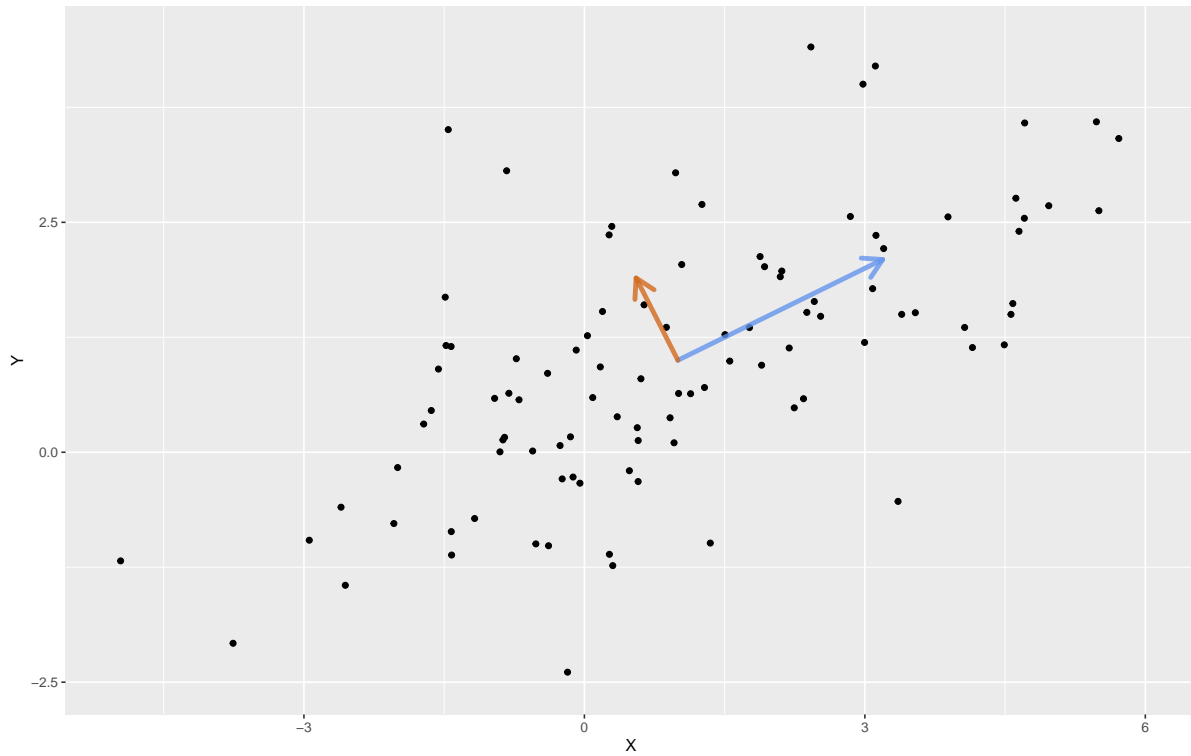
Figure 1.4: The dataset with the first loading vector (blue) and the second loading vector (orange) superimposed. Each loading vector has been rescaled by the square root of the corresponding eigenvalue, to give an indication of the variability in that direction.

To compute the principal components directly via R, we can use the `prcomp` command, as shown below. This is generally the preferred option over computing the eigenvalue decomposition by hand: `prcomp` is more flexible (allowing one, for example, to scale the data before computing the principal components) and is also numerically more stable.[1] For our simple dataset, the end result is the same:

```
pca <- prcomp(df)
pca
```

```
Standard deviations (1, .., p=2):
[1] 2.44949 1.00000

Rotation (n x k) = (2 x 2):
         PC1          PC2
X 0.8944272 -0.4472136
Y 0.4472136  0.8944272
```

> ⚠️ **Warning**
>
> Note that `prcomp` returns (among other things) the standard deviations, which are the square roots of the variances (eigenvalues). To compare the output of `prcomp` with the results of the eigenvalue analysis, **make sure to take the square of the standard deviations**, and you will see the eigenvalues appear:
>
> ```
> pca$sdev^2
> ```
>
> ```
> [1] 6 1
> ```

1.2.4 Standardizing the predictors

Prior to doing principal component analysis, the data are often standardized by subtracting the mean for each feature and dividing by the standard deviation. If the original predictors in our dataset are given by $\mathbf{X}_i$, $i = 1, \ldots, D$, this means that we introduce standardized variables $\mathbf{Y}_i$, given by

$$\mathbf{Y}_i = \frac{\mathbf{X}_i - \bar{\mathbf{X}}_i}{\sqrt{S_i^2}},$$

---

[1]R has two commands to compute the principal components: `prcomp` and `princomp`. The former computes principal components using the so-called singular value decomposition (SVD) and is preferred for numerical stability. The latter, `princomp`, uses the eigenvalue decomposition as is provided for backwards compatibility with SAS.

where S_i^2 is the variance of $\mathbf{X}_i$. The resulting variables $\mathbf{Y}_i$ will have mean 0 and variance 1.

Standardizing the predictors means that they will be comparable in magnitude: variables whose variance is small will gain in importance and large variables will decrease in importance, roughly speaking. This may change the PCA output significantly!

As a rule of thumb, you should standardize variables that are measured in different units (e.g. seconds, meters, Watt, ...), since the unit can be rescaled without affecting the physical meaning of the variable, or its relation to other variables (e.g., rescaling a variable expressed in meters by a factor of 1000 is the same as expressing that variable in kilometers). By contrast, variables that are measured in the same units or that are unitless should not be rescaled (or should be rescaled collectively).

For an example of the latter, think about pixel intensities in an image dataset (such a dataset is in fact analyzed in Section 2.2). Pixels near the edge of the image presumably are part of the background, don't vary that much, and are relatively unimportant, whereas pixels in the center are likely to be part of the image subject, vary a lot, and carry a lot of information. Standardizing the pixels would make the pixels near the edge just as variable as the pixels in the center, which would greatly amplify the noise in the image at the expense of the useful information in it!

Standardizing the predictors is also referred to as *scaling*.

1.3 Interpreting the PCA results

In this section we will discuss a number of useful results that follow from PCA. We will use the bodyfat dataset as an illustration throughout.

```
pc <- prcomp(bodyfat_predictors)
pc
```

```
Standard deviations (1, .., p=3):
[1] 7.2046011 3.7432587 0.1330841

Rotation (n x k) = (3 x 3):
                                PC1         PC2         PC3
triceps.skinfold.thickness 0.6926671   0.1511979   0.7052315
thigh.circumference        0.6985058  -0.3842734  -0.6036751
midarm.circumference       0.1797272   0.9107542  -0.3717862
```

We can also ask R to print a summary of the principal component analysis for us. This will give us a table with the proportion of variance explained by each principal component, as well

as the cumulative proportion (the amount of variance retained by that principal component and all previous ones).

```
summary(pc)
```

```
Importance of components:
                          PC1    PC2     PC3
Standard deviation     7.2046 3.7433 0.13308
Proportion of Variance 0.7872 0.2125 0.00027
Cumulative Proportion  0.7872 0.9997 1.00000
```

1.3.1 The score plot

Perhaps the most useful visualization of the PCA is the so-called **score plot**, which is nothing but a scatter plot of the first two principal components. It often happens that the score plot is sufficient to discern patterns in the data, such as clusters.

Figure 1.5 shows a score plot for the bodyfat dataset. While no obvious patterns in this dataset stand out, the plot does show that the principal components are uncorrelated, and this is a good confirmation of what we already know on theoretical grounds. It is customary to put the percentages of variance explained on the axis labels of the score plot, so that the person interpreting it can have an idea of how well the first two principal components describe the data.

As an aside, above we noted that the principal components are uncorrelated with one another. We can also verify that this is the case numerically. The result is not exactly zero, but it is very small:

```
cov(pc$x[,1], pc$x[,2])
```

```
[1] -2.604824e-16
```

1.3.2 The loadings plot

A **loadings plot** shows the relations of the original variables and the principal components. This is often a useful visualization to see how each variable contributes to the principal components. Moreover, one can show that the loadings are proportional to the Pearson correlations between the principal components and the variables. Hence, if a loading is positive (negative), that variable will be positively (negatively) correlated with that principal component.[2]

---

[2]There is a closely related plot type, the profile plot, which differs from the loadings plot in that it has the Pearson correlations on the y-axis. Otherwise the two plot types are identical.
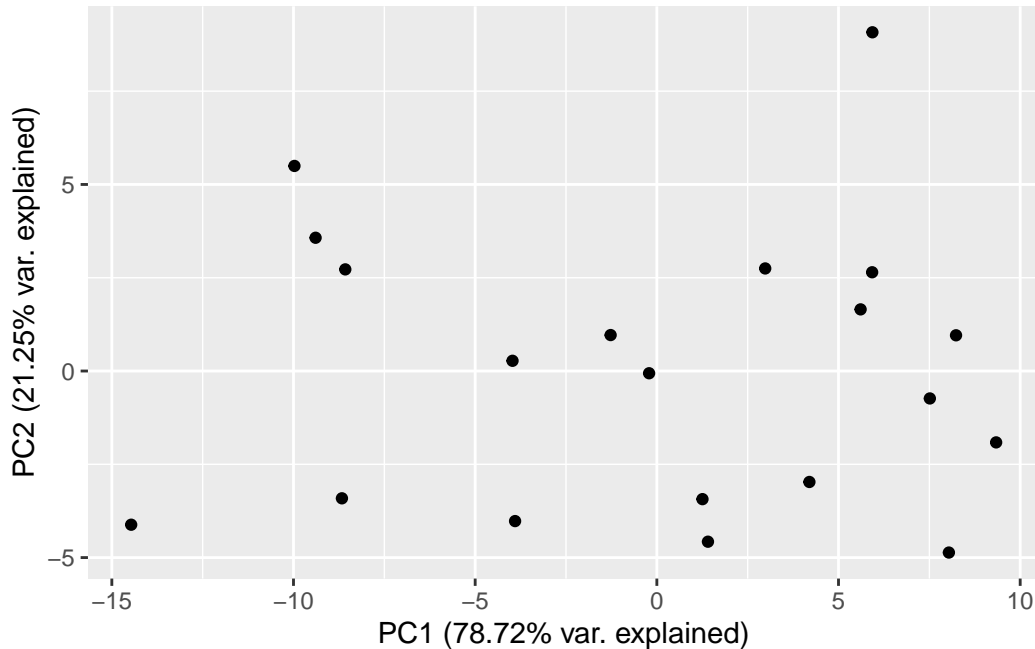
Figure 1.5: Score plot for the body fat dataset.

At the beginning of this chapter we hypothesized that the variables `thigh.circumference` and `triceps.skinfold.thickness` would contribute about equally to the first principal component. From Figure 1.6 we see that this is indeed the case: both variables have loadings approximately equal to 0.7, when we consider the first principal component. We also see that the second principal component is mostly made up of the variable `midarm.circumference`.

Unfortunately there is no command in base R or ggplot to create a loadings plot – you have to make one yourself.

1.3.3 The number of principal components

We now know how to calculate the D principal components for a given D-dimensional dataset, and we've seen that the principal components correspond to the directions in which the dataset varies the most. The real power of PCA, and the reason why it is so ubiquitous in data analysis, is that we can now selectively discard principal components that are not informative. By doing this, we obtain a dataset with fewer features, which is hence easier to analyze, and where the discarded features do not contribute too much to the expressivity of the data. This is what makes PCA into a *dimensionality reduction* method.

The question remains what principal components to discard. There are no universally accepted rules for this, but there are a couple of rules of thumb that can help us make an informed
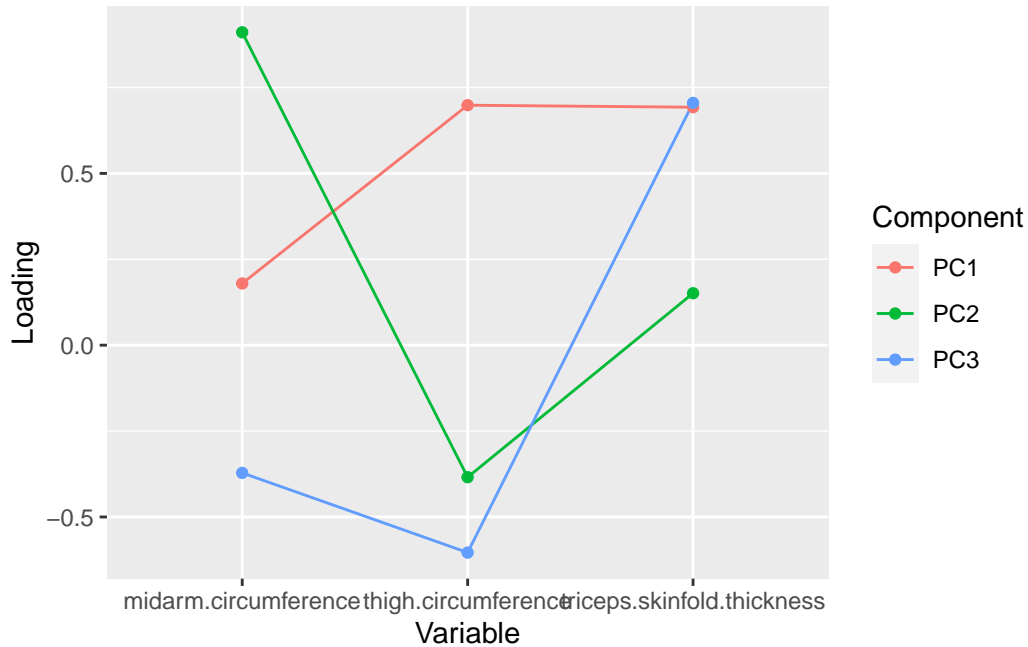
Figure 1.6: The loadings plot shows how the original variables are related to the principal components.

choice. Most of these rules take into account the total amount of variance retained by the first K principal components, defined as

$$S_K = \frac{\sum_{i=1}^{K} \lambda_i}{\sum_{j=1}^{D} \lambda_j},$$

Recall that the total amount of variance can be computed directly within R by using the `summary` command (and look for the "Cumulative Proportion" row):

```
summary(pc)
```

```
Importance of components:
                          PC1    PC2     PC3
Standard deviation     7.2046 3.7433 0.13308
Proportion of Variance 0.7872 0.2125 0.00027
Cumulative Proportion  0.7872 0.9997 1.00000
```

The number K of principal components can be chosen so that a fixed amount of variance (for example, 95%) is retained. To see this idea in action, let's apply it to the body fat dataset.

The relative amount of variance explained by each principal component can then be calculated as follows:

```
var_explained <- pc$sdev^2 / sum(pc$sdev^2)
var_explained
```

[1] 0.787222422 0.212508963 0.000268615

and the total amount of variance explained cumulatively by

```
total_var <- cumsum(var_explained)
total_var
```

[1] 0.7872224 0.9997314 1.0000000

Note that these numbers agree with the output of the `summary` command. For what follows, it will be easier to have access to these quantities as straightforward R vectors.

We see that the first two principal components explain 78.7% and 21.3% of the total variance, respectively, and together they explain more than 99.97% of variance in the dataset. It therefore seems reasonable to discard the last principal component, which contributes less than 0.03% of variance.

For datasets with many features, a **scree plot** or **elbow plot** can be helpful to identify high-variance principal components. In a scree plot, the amounts of variance are plotted in descending order, so that one can identify at a glance the amount of variability contributed by each principal component. Some scree plots also include the total amount of variability, S_K, as a function of K.

R can make a pretty basic scree plot for you, via the screeplot command.

```
screeplot(pc, main = "Principal components", type = "lines")
```
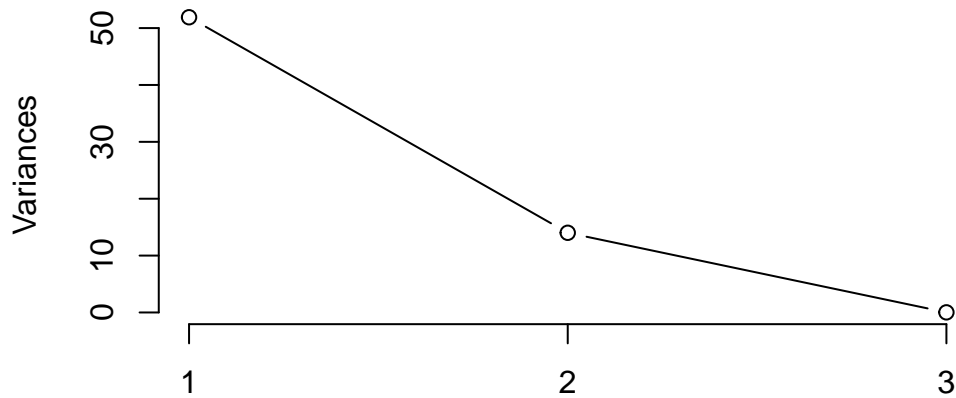
Principal components

Figure 1.7: A scree plot made with the base R `screeplot` command.

With a bit more work, you can build your own scree plot, which is often preferable if you want to customize the plot, to include for example the cumulative variance, as in figure Figure 1.8.
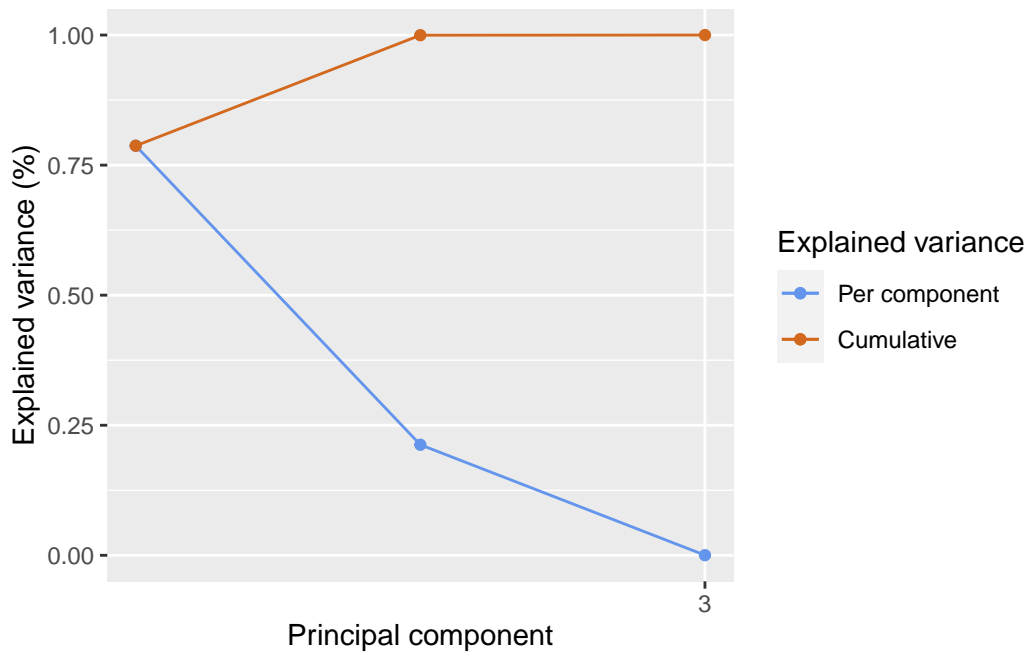

Figure 1.8: A scree plot for the body fat dataset confirms that the first two principal components explain almost all of the variance in the data.

Scree plots are often also referred to as "elbow plots", since the plot typically (but not always) shows an "elbow" where the explained variance levels off. However, spotting the exact location

of the elbow is very subjective, and it is typically more appropriate to take the point where the remaining principal components only contribute some small percentage of variation, for example 5%.

1.3.4 Biplots

The **biplot** consists of a loadings plot overlaid on a score plot. By default, it shows the first two principal components as a scatter plot, together with a set of vectors, one for each original variable, showing the contribution of that variable to the first two principal components. Biplots are relatively complex, but it is worth understanding what they encode.
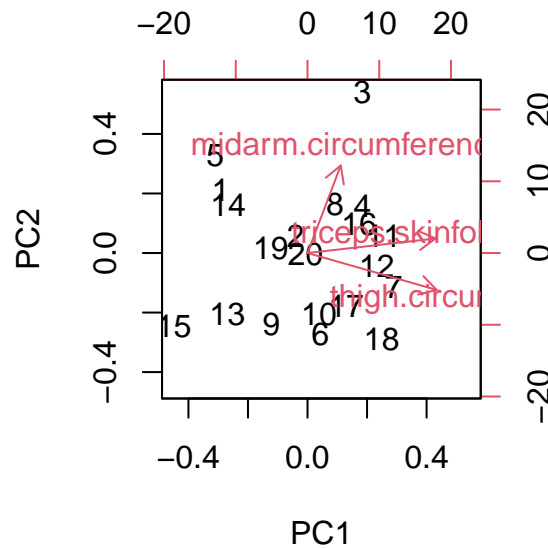
```
biplot(pc)
```


Figure 1.9: The biplot provides information about the transformed data and the loadings.

The numbers on the plot represent the data points from the original dataset, expressed relative to the first two principal components. This is the part of the biplot that is like a score plot. The red arrows, on the other hand, represent the original variables in the dataset (as shown by the labels attached to them) and are expressed on the top and right-most axis, which show how much each variable contributes to the first two principal components. The red arrows carry the same information as a loadings plot (in a different form), when you consider only the first two principal component.

At one glance, we see that `triceps.skinfold` and `thigh.circumference` are the most important contributors to the first principal component, and that the second principal component

21

is almost entirely made up by `midarm.circumference`. This confirms our intuition from the beginning of this chapter, as well as the conclusions that we drew from the loadings plot.

1.4 Principal component regression

Once we have performed PCA, we can build a linear model using the principal components that we have chosen to retain. This is known as *principal component regression.* Let's see this in action for the bodyfat dataset. We start from a model where we include the first principal component only:

```
pc1 <- pc$x[, "PC1"]
outcome <- bodyfat$bodyfat

model_1 <- lm(outcome ~ pc1)
summary(model_1)
```

```
Call:
lm(formula = outcome ~ pc1)

Residuals:
    Min      1Q  Median      3Q     Max
-5.1357 -1.8821  0.2682  1.7107  3.4992

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 20.19500    0.58688  34.411  < 2e-16 ***
pc1          0.61366    0.08358   7.343 8.13e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.625 on 18 degrees of freedom
Multiple R-squared:  0.7497,    Adjusted R-squared:  0.7358
F-statistic: 53.91 on 1 and 18 DF,  p-value: 8.128e-07
```

Adding the second principal component results in a bigger model, but the coefficient of the second principal component is not significant. We therefore do not include it in our final model and we stay with `model_1`.

```
  pc2 <- pc$x[, "PC2"]
  model_12 <- lm(outcome ~ pc1 + pc2)
  summary(model_12)
```

```
Call:
lm(formula = outcome ~ pc1 + pc2)

Residuals:
    Min      1Q  Median      3Q     Max
-3.9876 -1.8822  0.2562  1.3209  4.0285

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 20.19500    0.56604  35.678  < 2e-16 ***
pc1          0.61366    0.08061   7.613 7.12e-07 ***
pc2         -0.23785    0.15514  -1.533    0.144
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.531 on 17 degrees of freedom
Multiple R-squared:  0.7801,    Adjusted R-squared:  0.7542
F-statistic: 30.15 on 2 and 17 DF,  p-value: 2.564e-06
```

In this way, we obtain a simple least squares model, with one predictor and one outcome. Quite a simplification from our original linear model, which had three predictors and suffered from multi-collinearity. This reduction from 3 to 1 predictors may not seem like a big deal, but in the next chapter we will see examples of how PCA can be used to reduce datasets with many hundreds of variables to just 2-3 components, while maintaining the essential information in the dataset.

In this section we have built the linear model "by hand", but in reality you would probably not want to do this, for two reasons:

1. Testing each predictor at a time when there are many hundreds of candidates to choose from is impractical.
2. Using the forward model building procedure, as we did above, comes with significant multiple-testing issues.

Luckily, there are R packages that will do the PCA, select the optimal number of components, and build the regression model all for us. Here we use the `pls` package. We tell it to consider all principal components, and to assess the error through cross-validation. The details of this

will be discussed in class, but the important line in the output is labeled `RMSEP`. This shows a measure of error in function of the number of components taken into account. We see that this error is the lowest when only the first principal component is considered.

```r
library(pls)

pcr_model <- pcr(bodyfat ~ ., data = bodyfat, validation = "CV")
summary(pcr_model)
```

```
Data:    X dimension: 20 3
    Y dimension: 20 1
Fit method: svdpc
Number of components considered: 3

VALIDATION: RMSEP
Cross-validated using 10 random segments.
       (Intercept)  1 comps  2 comps  3 comps
CV            5.239    2.612    2.597    2.681
adjCV         5.239    2.602    2.583    2.656

TRAINING: % variance explained
         1 comps  2 comps  3 comps
X          78.72    99.97   100.00
bodyfat    74.97    78.01    80.14
```

There are various rules to select the optimal number of components to consider. One such rule is the "1-sigma rule" (Hastie, Tibshirani, and Friedman 2009) which considers the model with the least number of components whose cross-validation error is at most one standard deviation away from the optimal model. In our case, this again results in a model with only one predictor.

RMSEP

4.5
3.5
2.5

0.0   0.5   1.0   1.5   2.0   2.5   3.0

Number of components

Abs. minimum
Selection
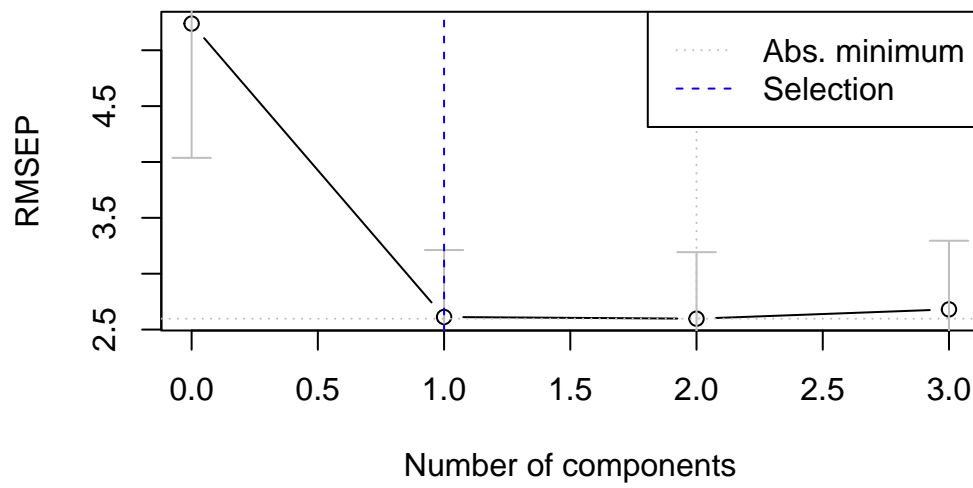
[1] 1

2 Applications of principal component analysis

Principal component analysis is often a necessary first step when there are a large number of independent variables that need to be analyze simultaneously. Many devices in a modern lab produce this kind of high-dimensional data: for example, a reading for a single sample obtained via gas chromatography-mass spectrometry (GC-MS) or hyperspectral imagining (HSI) is a vector with 100s of entries, and with the number of samples often running in the 100s as well, we need a technique like PCA to find the needle in the haystack.

In this chapter we consider a number of real-world examples, from the life sciences and beyond, where PCA and dimensionality reduction prove to be essential.

> ⚠️ Warning
>
> Given that the examples in this chapter deal with real-world data, which is often large and messy, the R code is at times a bit more complex than in the previous chapter. The underlying principles remain the same, however.

2.1 Adulteration of olive oil

This case study is a simplified version of an analysis done by the group of Prof. Van Haute of the Centre for Food Chemistry and Technology at GUGC to determine the adulteration of extra-virgin olive oil through the use of hyperspectral imaging data. The full analysis can be found in (Malavi et al. 2023). My thanks go to Prof. Van Haute for making the data available and for in-depth discussions regarding the analysis.

2.1.1 Problem statement

Extra-virgin olive oil (EVOO) is a type of oil that is made by cold-pressing the fruit of the olive tree without the use of chemicals or heat. It is considered the highest quality and most flavorful type of olive oil and is widely used in cooking and as a dressing for salads and other dishes. Extra-virgin olive is packed with antioxidants and healthy fats, making it a popular choice among health-conscious consumers. Due to its high quality and health benefits, extra-virgin olive oil is often more expensive than other types of olive oil.

Extra-virgin olive oil is sometimes adulterated, either accidentally, or deliberately, by the addition of other, cheaper vegetable oils. This is misleading to the customer and can pose health risks, for example through the introduction of allergens. As a result, manufacturers and food safety agencies have an interest in determining whether a given EVOO sample has been adulterated, and if so, to what degree.

One way to determine the chemical composition of an oil sample is through hyperspectral imaging (HSI). A hyperspectral imaging system will shine infrared light onto the sample and measure the reflection off the sample at different wavelengths. We will not go into the details of how this signal is acquired, but what is important is that the system outputs for each sample a so-called *spectral curve* describing the average reflectance at different wavelengths. By inspecting the spectral curve, we can establish whether the sample absorbs light of a specific wavelength, and this can point towards the presence of particular chemical structures that characterize the sample. An example of a spectral curve is shown in figure Figure .

In our case study we want to determine whether hyperspectral imaging can be used to detect the adulteration of extra-virgin olive oil. More precisely, we have the following research questions:

1. *Can hyperspectral imaging be used to detect whether olive oil has been adulterated with other kinds of oils?*
2. *If so, can the amount of adulteration be quantified (e.g. as a percentage)?*

To investigate these research questions, Malavi et al. (2023) acquired readings from 13 different kinds of unadulterated EVOO, together with readings from 42 adulterated mixtures. Each adulterated mixture was prepared by taking olive oil and adding one of 6 different vegetable oils at 7 different percentages (ranging from 1% to 20% adulterant). Each sample was prepared and imagined in triplicate, resulting in 183 spectra. Each spectrum is a vector of length 224, describing the reflectance at 224 equally distributed wavelengths from 700 to 1800 nm.

Below, we read in the dataset and we print out the first 6 columns of a random sample of 10 rows (the full dataset is too large to display in its entirety). Note that the dataset has 228 columns: 4 of these are metadata variables described below and the remaining 224 columns describe the spectrum for each sample. The metadata variables are:

- `Sample ID/Wavelength`: The name of the oil or mixture, and the mixture ratio (if applicable)
- `Sample`: A unique integer identifying each sample. Not used in the subsequent analysis.
- `Classification`: Whether the sample is primarily olive oil or not.
- `% Adulteration`: The percentage of food oil added to the mixture. For pure EVOO this is 0, while for pure food oil it is 100%. For the other mixtures, it is one of 1%, 2%, 4%, 8%, 12%, 16%, or 20%.

```
# A tibble: 10 x 6
   `Sample ID/Wavelength`      Sample Classification % Adulte~1 938.9~2 942.4~3
```

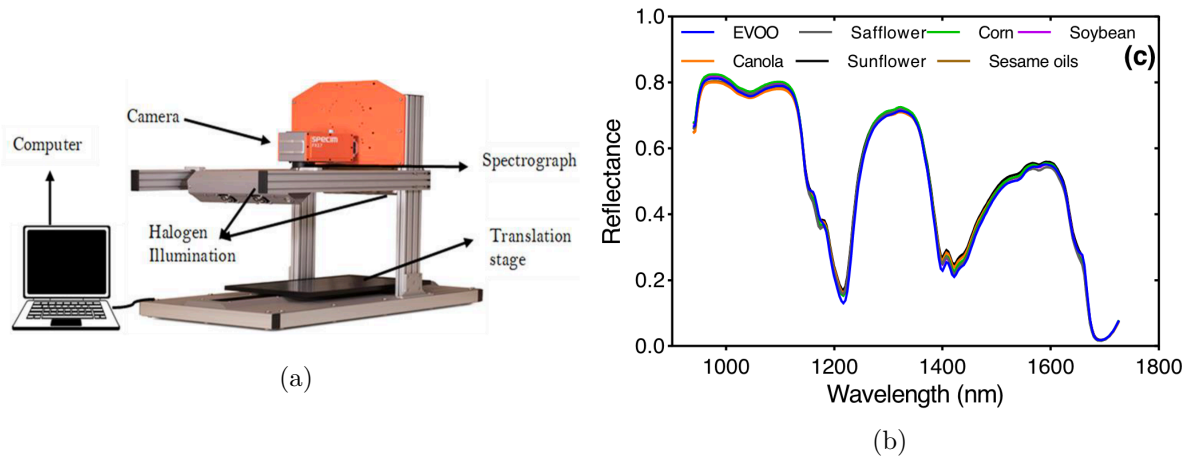Figure 2.1: Hyperspectral imaging system (left) and typical output spectra (right). Figure source: Malavi et al. (2023).

```
   <chr>                         <dbl> <chr>              <dbl>  <dbl>  <dbl>
 1 EVOO flavored with Chilli        28 Olive                 0  0.638  0.643
 2 Olive Oil/Corn Oil 99/1          80 Olive                 1  0.649  0.655
 3 Olive Oil/Sunflower Oil 96/4    150 Olive                 4  0.659  0.664
 4 Olive Oil/Soybean Oil 99/1      101 Olive                 1  0.609  0.613
 5 Olive Oil/Soybean Oil 92/8      111 Olive                 8  0.632  0.636
 6 Olive Oil/Canola Oil 84/16      137 Olive                16  0.652  0.657
 7 Olive Oil/Canola Oil 88/12      133 Olive                12  0.655  0.660
 8 Olive Oil/Sesame Oil 98/2       166 Olive                 2  0.652  0.657
 9 Olive Oil/Sunflower Oil 99/1    144 Olive                 1  0.652  0.657
10 Olive Oil/Canola Oil 92/8       132 Olive                 8  0.654  0.660
# ... with abbreviated variable names 1: `% Adulteration`,
#   2: `938.94000200000005`, 3: `942.45001200000002`
```

2.1.2 Inspecting the spectral plots

As a first step in our analysis, let's compare the typical spectrum of EVOO with that of the other 6 vegetable oils. We see that overall the spectra are quite similar (they are all oils, after all) but that there are small differences between the different kinds of oil. On the other hand, if we are given a new, unlabeled spectrum, it would be quite difficult to "guess" just by looking what type of oil it is. This is where dimensionality reduction will help us!
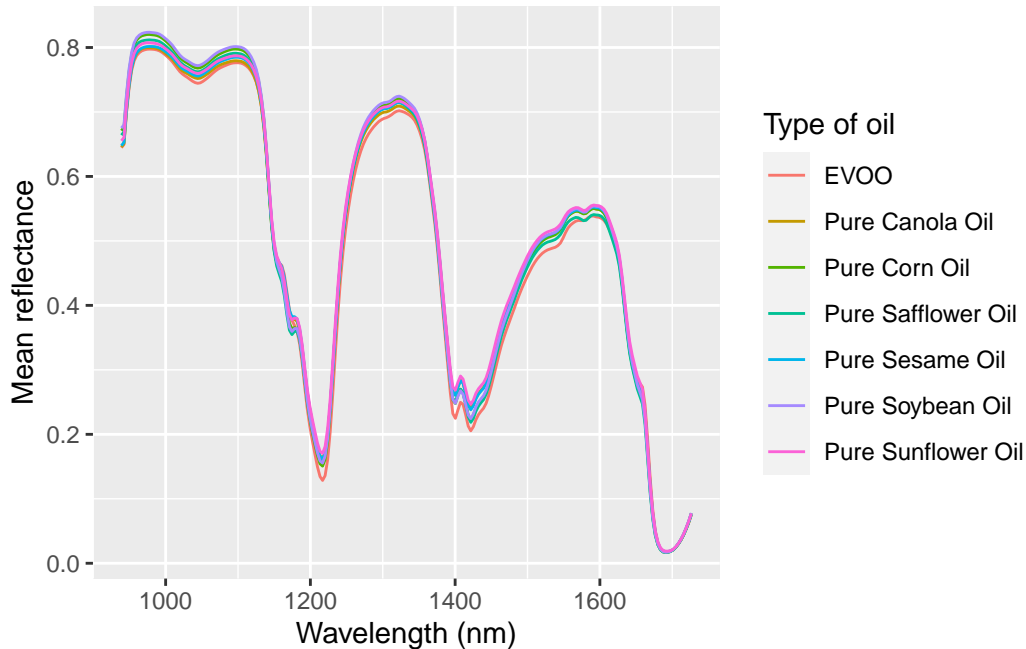
Figure 2.2: Typical spectra of EVOO and different types of vegetable oils. Note the similarity with figure Figure 2.1b.

2.1.3 Computing the principal components

We are now in a position to compute the principal components, and to figure out what they tell us about olive and other oils. Two issues to keep in mind:

1. Our dataset contains both metadata and spectral data: we want to make sure to compute the principal components only for the columns containing spectral data!
2. For spectral data, we definitely do **not** scale the different variables!

```
pca_oils <- oils %>%
  select(all_of(spectra)) %>%
  prcomp(scale = FALSE)
```

The first two principal components explain more than 90% of the variance in the data, as shown below.

The loadings for the first two principal components tell us something about what the spectra look like overall, and what variations between the individual spectra look like. This is of course an extremely coarse perspective, given that we only consider two principal components.

We started this section with the observation that there are minute differences between the spectra of different oils, but we admitted that it would be quite difficult to tell two spectra
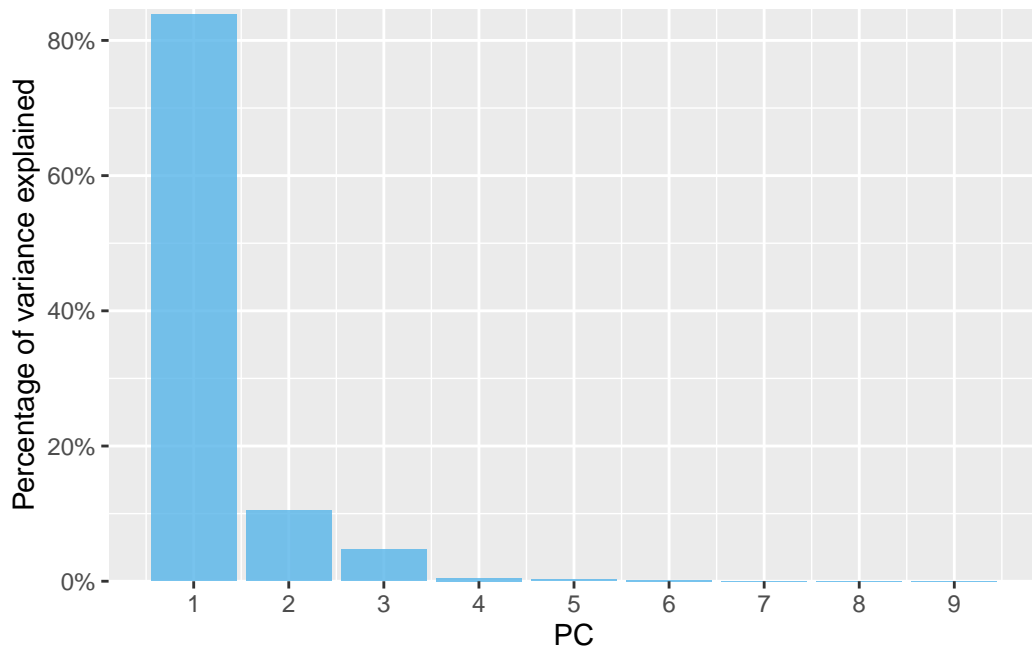
Figure 2.3: Percentage of variance explained by the first 10 principal components. The first two principal components explain 94% of the variance in the data.
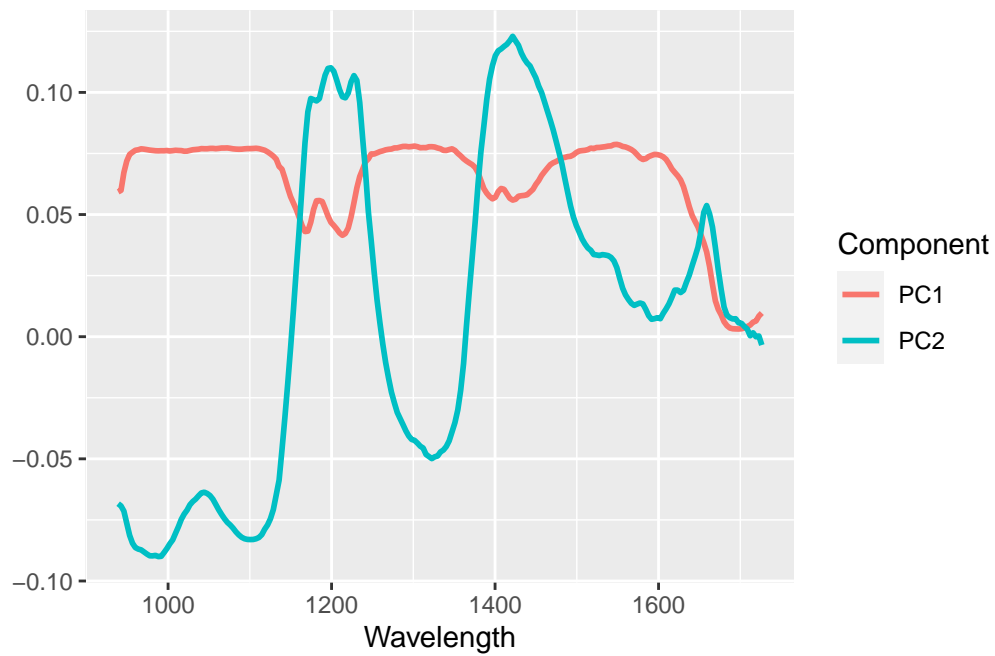

Figure 2.4: Loadings for the first two principal components.

apart just by eye. We are now in a position where we can solve this issue: by considering only the first few principal components we get a reduced number of variables that (hopefully) can be used to tell the spectra apart.

Below we have a scatterplot of the first two principal components. The EVOO samples are round dots, and the adulterated oils are triangles colored by the percentage of adulteration. The EVOO samples are clearly distinct from the adulterated oils (they are separated by the second principal component). This answers our first research question with a "yes": we can clearly tell pure and adulterated oils apart.
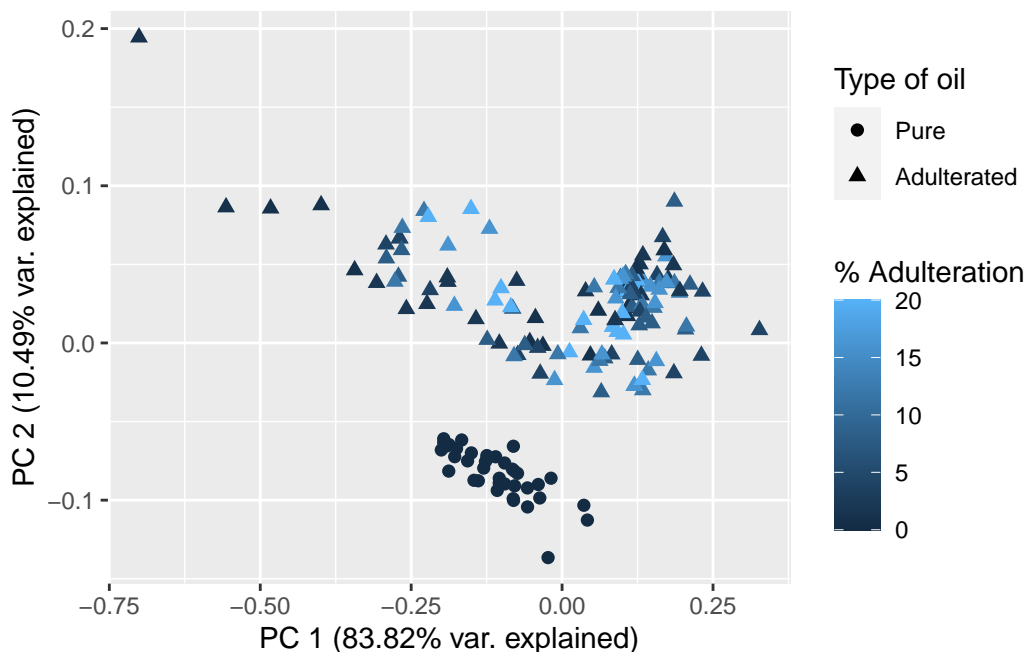

Figure 2.5: Principal component plot for the EVOO and adulterated oil samples. Pure vegetable oil samples are not shown.

2.1.4 Predicting the percentage of adulteration

Now that we are able to tell EVOO and adulterated samples apart, it is time to consider our second research question: given an adulterated oil sample, can we predict the percentage of vegetable oil that was added? Looking at Figure 2.5, it looks like we'll need more than two principal components: we see no clear pattern in the percentage adulteration that we or a linear model could exploit.

Before building our model, we need to do some data preparation. We select from our dataset only those rows that contain adulterated oils, and we split the resulting dataset into a *training dataset* (containing 80% of the data) and a *test dataset* (containing the remaining 20%). The

idea is that we set apart the test dataset and build our model using only the training dataset. The test dataset is used to evaluate how well the model performs on data that it has never seen before. Using the full dataset to build the model and evaluate it would result in an overly optimistic estimate of the predictive capabilities of our model.

```
Number of rows in the training dataset: 101
```

```
Number of rows in the test dataset: 25
```

We could build a linear model with more than two components by hand, just like we did for the bodyfat dataset in the previous chapter. However, it is easier to let R do the work for us, and we will use the the `pls` package for this. This package has a number of advantages: you can use the same formula syntax that you know from the `lm` command, and the package comes with a number of different regression models. Here we build a principal component regression model and a partial least squares regression model.

```r
library(pls)

# Principal component regression
pcr_model <- pcr(
  `% Adulteration` ~ ., data = adulterated_train, scale = FALSE, validation = "CV", ncomp
)

# Partial least squares regression
pls_model <- plsr(
  `% Adulteration` ~ ., data = adulterated_train, scale = FALSE, validation = "CV", ncomp
)
```

The argument `validation = "CV"` will cause the model to select by itself the appropriate number of components to use, up to a maximum of 10 components (this is controlled by the `ncomp = 10` argument). To do this it will use the so-called cross-validation (CV) strategy: it will repeatedly split the data into a training and a validation set (not to be confused with the training and test set that we created ourselves above) and use the performance on the validation set to select the appropriate number of components.

Now that we have two models, we can ask them to make predictions on the held-out test data, and we can compare the predictions made between both models. In the prediction plot below we see in general good agreement between the actual percentage of adulteration (on the x-axis) and the predicted percentage (on the y-axis). Both models perform comparably well, at least as far as can be judged from the plot.

```
pcr_pred <- predict(pcr_model, adulterated_test, ncomp = 10)
pls_pred <- predict(pls_model, adulterated_test, ncomp = 10)
```
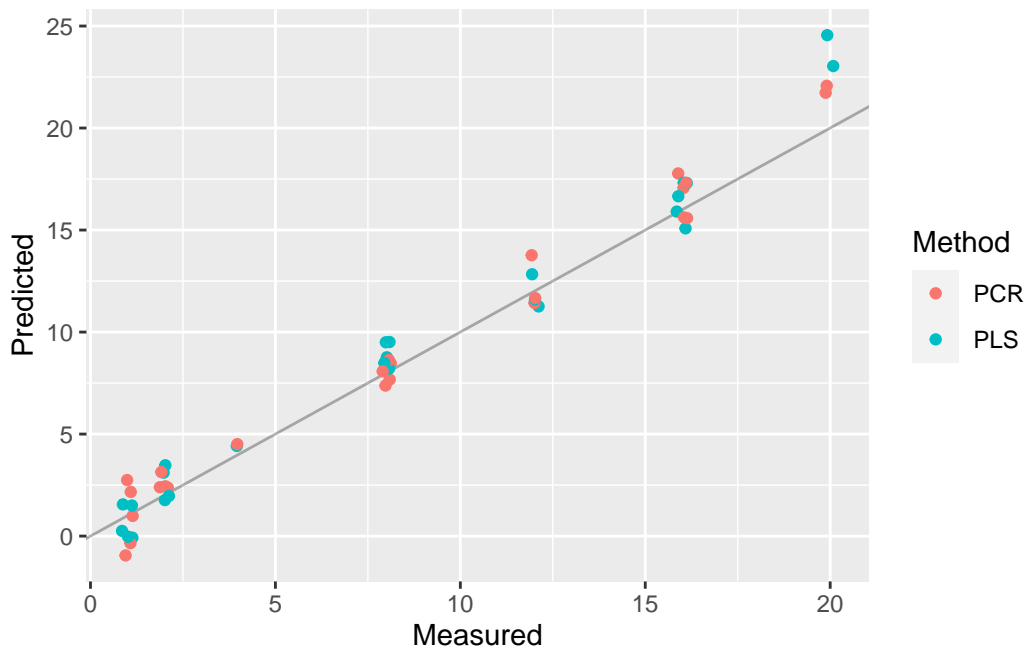

2.1.5 A cautionary note

In our model building procedure we have glossed over a few critical steps that may cause doubt on the validity of the model or the conclusions that we can draw from our analysis.

- Chiefly among these is the splitting into training and test data. We did that correctly to train our model, but everything that came before it (including the exploratory data analysis) used the entire data set. To build a model that is not overly optimistic, you should set aside some data to serve as test data at the beginning of your analysis, and then use only the training data to determine the appropriate number principal components, to classify oils into adulterated and pure, and so on. *Your model should never "see" the test data, until the very end.*

- We have trained and evaluated our model only on adulterated oils. In reality, however, we will present the model with unknown oil samples, which may or may not be adulterated, and the performance on such samples is an open question. A better model could be built by including EVOO samples in our dataset, or by first classifying oils into adulterated/pure. Each approach comes with a number of complications that would lead us too far.

- Our model willfully ignores a number of important variables in the dataset. For example, we don't take into account the kind of vegetable oil that was used to adulterate a sample. By distinguishing the type of adulterant, we could presumably build a better model, but this too would make the analysis significantly more complex.

A more complete model that addresses all of these concerns (and more) can be found in (Malavi et al. 2023).

2.2 Eigenfaces

Our last example is not life sciences based, but serves as an illustration to show that PCA is a powerful technique in data analysis, which can be used to reduce the number of degrees of freedom in a large dataset.

We use the Olivetti dataset of human faces, which contains 400 frontal photographs of human faces. Each face is a grayscale image of 64 by 64 pixels, where the intensity of each pixel is a value between 0 (completely black) to 255 (completely white). Each image can be represented as a 64×64 matrix, but it will be more convenient to take the columns of this matrix and lay them out one after the other to obtain a vector with $64 \times 64 = 4096$ entries, as in Figure 2.6.
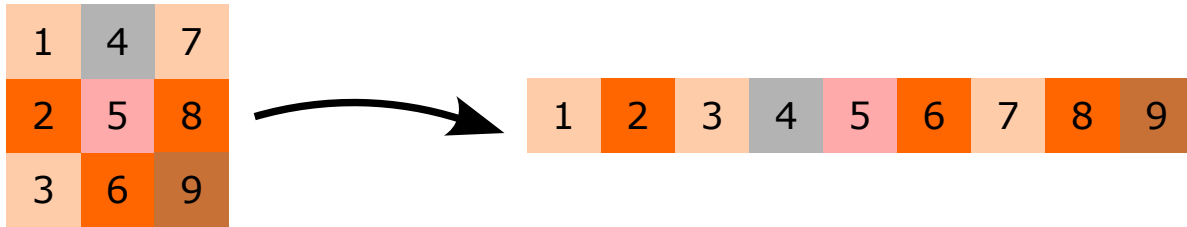

Figure 2.6: An image that is N pixels high and M pixels wide can be viewed as a matrix with N rows and M columns, or as a vector with $N \times M$ elements. Here, N and M are both equal to 3.

First, we load the dataset. Note that the dataset comes as a data matrix with 4096 rows and 400 columns.

```
library(loon.data)
data(faces)
dim(faces)
```

```
[1] 4096  400
```

Each column in the data matrix represents a face, laid out as a column vector with 4096 as in Figure 2.6. We can assemble these vectors back into images and visualize them. This requires

some R commands that we haven't covered; you don't have to understand what this code does.


Figure 2.7: Six faces from the Olivetti dataset.

Doing a principal component analysis is a simple matter of running `prcomp`. Despite the size of the dataset, this component should not take more than a second to run.

```
pc_olivetti <- prcomp(faces)
```

Note that there are 400 principal components in this dataset. We can visualize their relative importance via a scree plot, which we limit to the first 50 components for clarity, since the remaining 350 components contribute almost no variance. This indicates that we can probably discard most of the principal components without losing much of the expressivity of our dataset. We will see further down that this is indeed the case!
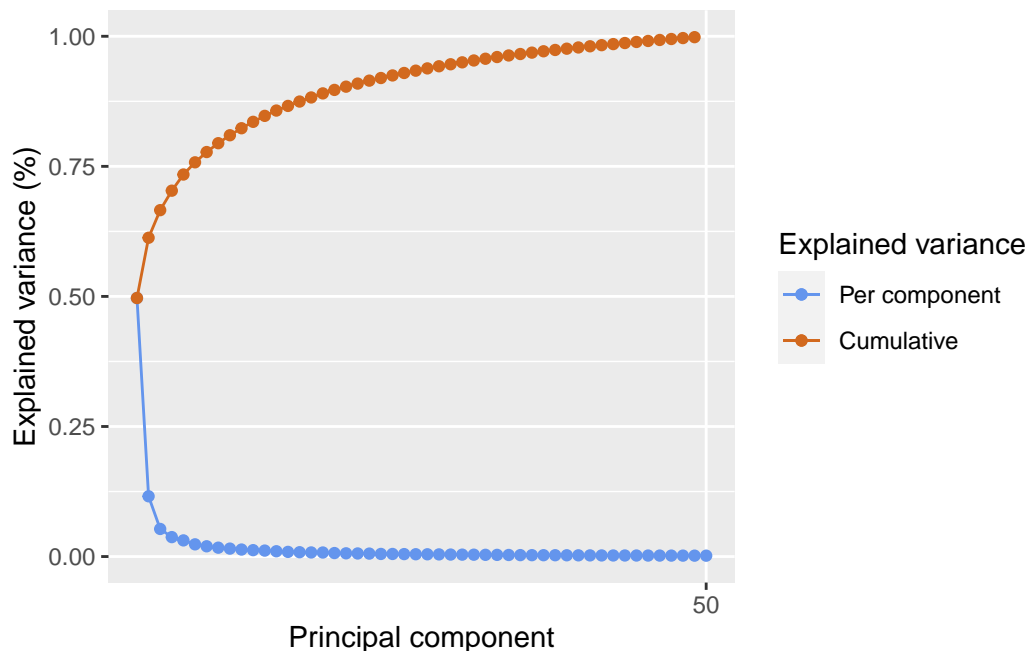
One of the advantages of the faces dataset is that the loadings vectors can be represented graphically, and that we can reason about them. Figure 2.8 shows the first 8 loadings vectors, represented as images. How should we interpret these images? Each loadings vector represents a particular *pattern* in the dataset of all faces: the first loadings vector, for example, captures the overall structure of a human face, while the second represents the illumination from right to left. Probably there were some photos in the dataset that were illuminated from the left or the right. Loadings vector three does the same for the top-down illumination, and loadings vectors four through eight capture particular patterns involving the eyes or the eyebrows. *By selectively "mixing" all 400 loadings vectors, we can recreate any face in the dataset.*

To finish, let's also investigate how well PCA performs as a data reduction method. By retaining only a limited number of principal components, we can build "compressed" versions of the images that contain less information but are comparable to the eye. Figure 2.9 shows two original faces from the dataset (left), together with compressed versions involving the first 10, 40, and 80 most significant principal components. The version that uses only 10 components is quite generic and it is difficult even to distinguish the male and female face. The version with 80 components, on the other hand, is very close to the original.

It is worth realizing the amount of data compression realized by using PCA. The original images had 4096 degrees of freedom, whereas the rightmost versions in Figure 2.9 are described by the coefficients of 80 loadings vectors, more than a 50-fold reduction in degrees of freedom! Clearly there are some visual artifacts that appear in the compressed versions, but the faces are clearly distinguishable, and it seems very reasonable at this point that a machine learning algorithm (for example, to classify the faces, or to do segmentation) could take these compressed images as input and still perform well.
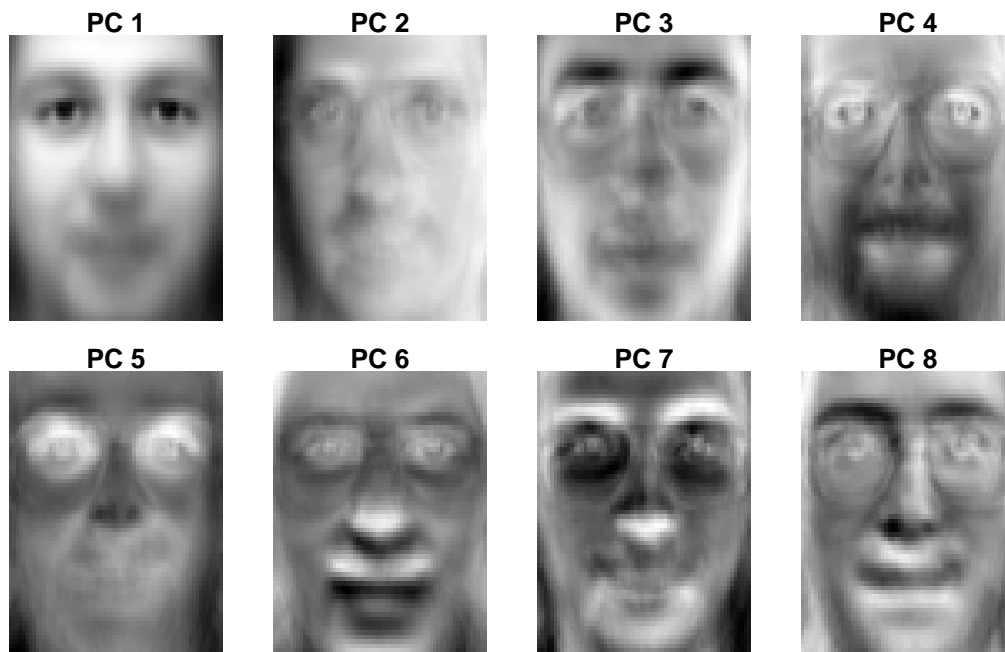
PC 1 **PC 2** **PC 3** **PC 4**

PC 5 **PC 6** **PC 7** **PC 8**

Figure 2.8: The first 8 loadings vectors of the Olivetti dataset represent particularly expressive patterns in the dataset.

Original **10 PCs** **40 PCs** **80 PCs**

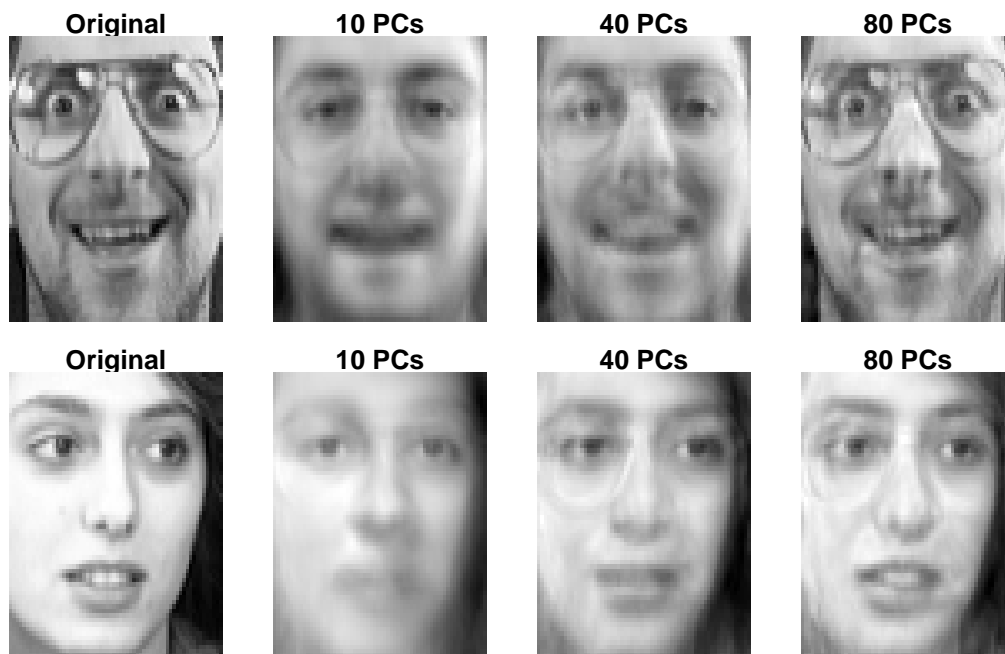**Original** **10 PCs** **40 PCs** **80 PCs**

Figure 2.9: Original images (left), and 3 PCA-reduced images with increasing numbers of principal components.

References

Bishop, Christopher M. 2006. *Pattern Recognition and Machine Learning.* Vol. 2. Information Science and Statistics. Springer, New York.

Hastie, T., R. Tibshirani, and J. H. Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer Series in Statistics. Springer.

Malavi, Derick, Amin Nikkhah, Katleen Raes, and Sam Van Haute. 2023. "Hyperspectral Imaging and Chemometrics for Authentication of Extra Virgin Olive Oil: A Comparative Approach with FTIR, UV-VIS, Raman, and GC-MS." *Foods* 12 (3): 429. https://doi.org/10.3390/foods12030429.

A Datasets

A.1 Point clouds with specific mean and covariance

The `rmvnorm` command, part of the mvtnorm package, provides a way to sample points from a multivariate normal distribution with a specific population mean and standard deviation. For example, the code snippet below generates 100 sample points from a distribution with mean μ and covariance matrix Σ given by

$$\mu = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \Sigma = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}. \tag{A.1}$$

The `mean` and `sigma` parameters are the *population* mean and covariance, and the *sample* mean and covariance will be slightly different:

```
bar_x <- colMeans(samples)
bar_x
```

```
[1] 0.9661279 0.7986743
```

```
sigma_x <- cov(samples)
sigma_x
```

```
         [,1]     [,2]
[1,] 2.162411 1.201452
[2,] 1.201452 3.173997
```

In some cases, we require a dataset whose sample mean and covariance are *exactly* equal to some given parameters. It turns out that we can achieve this by means of a judiciously chosen linear transformation. Assume that we have n datapoints $x_i \in \mathbb{R}^p$, and let A be an arbitrary $p \times p$ matrix and $b \in \mathbb{R}^p$ a vector. The transformed data points

$$y_i = Ax_i + b, \tag{A.2}$$

have sample mean $\bar{y}$ and covariance matrix Σ_y given

$$\bar{y} = A\bar{x} + b \quad \text{and} \quad \Sigma_y = A\Sigma_x A^T,$$

where $\bar{x}$ and Σ_x are the sample mean and covariance matrix of the x variables.

To make the sample mean equal to a given vector μ, and the sample covariance matrix equal to a given matrix Σ, we have to solve the following equations for A and b:

$$A\bar{x} + b = \mu \quad \text{and} \quad A\Sigma_x A^T = \Sigma.$$

Let's focus on the second equation first, since it only involves the matrix A. Since the covariance matrices Σ and Σ_x are positive definite, it turns out that there exist unique upper-triangular matrices R and R_x such that

$$\Sigma_x = R_x^T R_x \quad \text{and} \quad \Sigma = R^T R.$$

This is the so-called Cholesky decomposition; we don't have to worry about the details here, since R can compute R and R_x for us. Using this decomposition, the equation for A can be written as
$$A R_x^T R_x A^T = (R_x A^T)^T R_x A^T = R^T R,$$

so that it is sufficient to find a matrix A such that $R_x A^T = R$. This we know how to do: we just multiply both sides from the left by $(R_x)^{-1}$ and take the transpose to find

$$A = (R_x^{-1} R)^T.$$

With this expression for A, we can find b by setting

$$b = \mu - (R_x^{-1} R)^T \bar{x}.$$

We can code up these equations in R as in the snippet below. This approach works well for data points in low dimensions. For higher-dimensional data it is recommended (for reasons of numerical efficiency and stability) to avoid computing the inverse R_x^{-1} directly. Instead, we can solve a system of linear equations that yield the transformed points y_i directly. We will not pursue this alternative any further.

```
add_to_rows <- function(X, b) {
  t(apply(X, 1, function(row) row + b))
}

R <- chol(sigma)
R_x <- chol(cov(samples))
A <- t(solve(R_x) %*% R)
b <- mean - A %*% bar_x
```

```
samples_y <- samples %*% t(A)
samples_y <- add_to_rows(samples_y, b)
```

The resulting datapoints have mean and covariance exactly as given by Equation A.1:

```
colMeans(samples_y)
```

```
[1] 1 1
```

```
cov(samples_y)
```

```
      [,1] [,2]
[1,]    2    1
[2,]    1    3
```

Last, we can take a look at the relative location of the original and adjusted data points. We see that our procedure selectively moves points around to make the mean and covariance matrix equal to what we imposed on it.