ing the first d principal components (i.e., the matrix composed of the first d columns of $\mathbf{V}^T$), as shown in Equation 8-2.

Equation 8-2. Projecting the training set down to d dimensions

$$\mathbf{X}_{d\text{-proj}} = \mathbf{X} \cdot \mathbf{W}_d$$

The following Python code projects the training set onto the plane defined by the first two principal components:

```
W2 = V.T[:, :2]
X2D = X_centered.dot(W2)
```

There you have it! You now know how to reduce the dimensionality of any dataset down to any number of dimensions, while preserving as much variance as possible.

Using Scikit-Learn

Scikit-Learn's `PCA` class implements PCA using SVD decomposition just like we did before. The following code applies PCA to reduce the dimensionality of the dataset down to two dimensions (note that it automatically takes care of centering the data):

```
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)
X2D = pca.fit_transform(X)
```

After fitting the `PCA` transformer to the dataset, you can access the principal components using the `components_` variable (note that it contains the PCs as horizontal vectors, so, for example, the first principal component is equal to `pca.components_.T[:, 0]`).

Explained Variance Ratio

Another very useful piece of information is the *explained variance ratio* of each principal component, available via the `explained_variance_ratio_` variable. It indicates the proportion of the dataset's variance that lies along the axis of each principal component. For example, let's look at the explained variance ratios of the first two components of the 3D dataset represented in Figure 8-2:

```
>>> print(pca.explained_variance_ratio_)
array([ 0.84248607,  0.14631839])
```

This tells you that 84.2% of the dataset's variance lies along the first axis, and 14.6% lies along the second axis. This leaves less than 1.2% for the third axis, so it is reasonable to assume that it probably carries little information.