```
'Output':
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=1.0, n_estimators=50, random_state=None)

# print the feature importances
ada_boost_classifier.feature_importances_
'Output': array([0.  , 0.  , 0.58, 0.42])

# create a subset of data based on the relevant features
model = SelectFromModel(ada_boost_classifier, prefit=True)
new_data = model.transform(X)

# the irrelevant features have been removed
new_data.shape
'Output': (150, 2)
```

Resampling Methods

Resampling methods are a set of techniques that involve selecting a subset of the available dataset, training on that data subset, and using the remainder of the data to evaluate the trained model. Let's review the techniques for resampling using Scikit-learn. This section covers

- k-Fold cross-validation
- Leave-one-out cross-validation

k-Fold Cross-Validation

In k-fold cross validation, the dataset is divided into k-parts or folds. The model is trained using $k - 1$ folds and evaluated on the remaining kth fold. This process is repeated k-times so that each fold can serve as a test set. At the end of the process, k-fold averages the result and reports a mean score with a standard deviation. Scikit-learn implements K-fold CV in the module **KFold**. The module **cross_val_score** is used to evaluate the cross-validation score using the splitting strategy, which is **KFold** in this case.

Let's see an example of this using the k-nearest neighbors (kNN) classification algorithm. When initializing **KFold**, it is standard practice to shuffle the data before splitting.

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier

# load dataset
data = datasets.load_iris()

# separate features and target
X = data.data
y = data.target

# initialize KFold - with shuffle = True, shuffle the data before splitting
kfold = KFold(n_splits=3, shuffle=True)

# create the model
knn_clf = KNeighborsClassifier(n_neighbors=3)

# fit the model using cross validation
cv_result = cross_val_score(knn_clf, X, y, cv=kfold)

# evaluate the model performance using accuracy metric
print("Accuracy: %.3f%% (%.3f%%)" % (cv_result.mean()*100.0, cv_result.
std()*100.0))
'Output':
Accuracy: 93.333% (2.494%)
```

Leave-One-Out Cross-Validation (LOOCV)

In LOOCV just one example is assigned to the test set, and the model is trained on the remainder of the dataset. This process is repeated for all the examples in the dataset. This process is repeated until all the examples in the dataset have been used for evaluating the model.

```
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
```