Intro to scikit-learn

Michael Becker

PyData Boston 2013

Who is this guy?

Software Engineer @ AWeber
Founder of the DataPhilly Meetup group
@beckerfuffle
beckerfuffle.com

These slides and more @ github.com/mdbecker

Good morning everyone, My name is Michael Becker, I work in the Data Analysis and Management team at AWeber, an email marketing company in Chalfont, PA
I'm also the founder of the DataPhilly Meetup group
You can find me online @beckerfuffle on Twitter. At beckerfuffle.com, and I'm also mdbecker on github. I'll be posting the materials for this talk on my github.

On the shoulders of giants

- Machine Learning 101 tutorial from scikit-learn.

So I want to start this talk by thanking those who came before me.
None of the content from this talk is original. It's been influenced heavily by various other talks and resources around the web.
This talk is based primarily on the "Machine Learning 101" tutorial from the scikit-learn documentation.

On the shoulders of giants

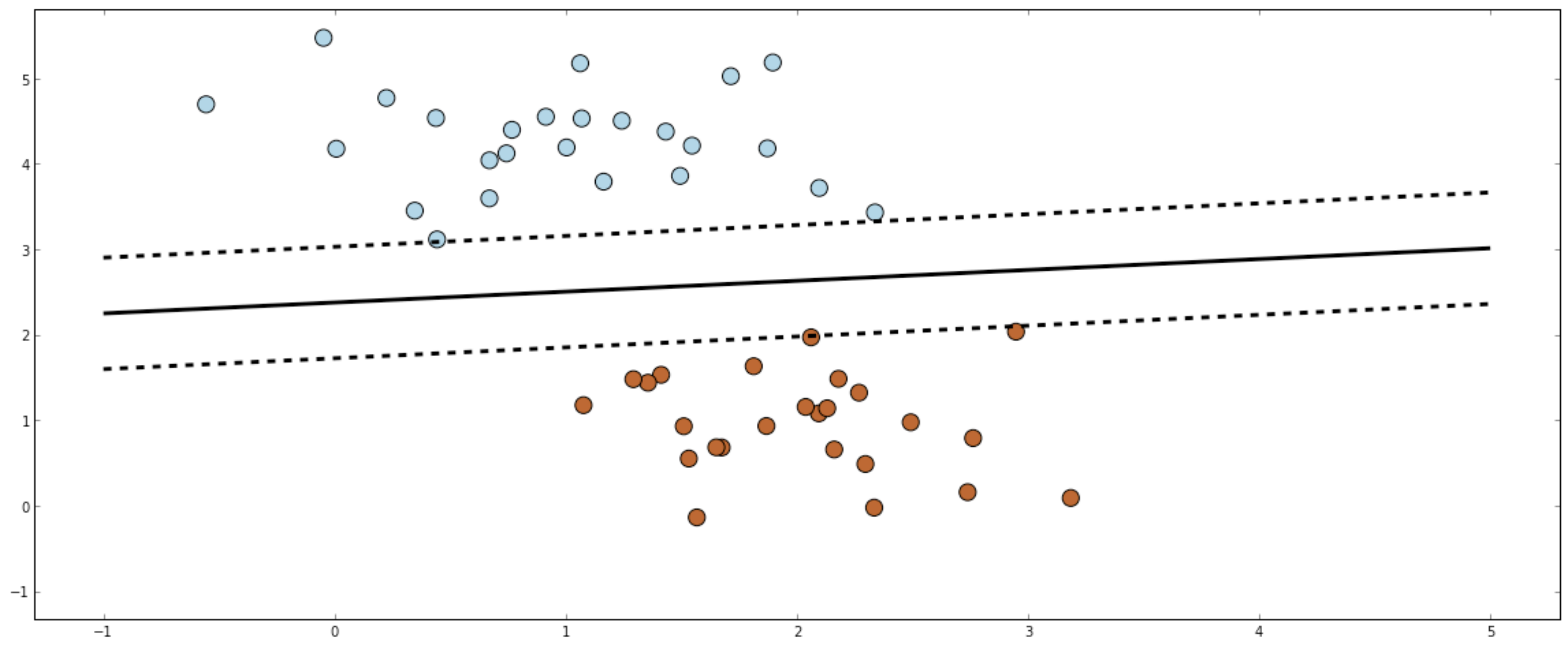
- Machine Learning 101 tutorial from scikit-learn.
- IPython notebooks from pycon 2013.

Additional thanks also to Jake Vanderplas for creating an excellent set of ipython notebooks for pycon 2013 which I've used for my code samples.
This talk will only cover a subset of what's available in these resources. I recommend you have a look at those to learn more about scikit-learn.

I'm not currently a contributor to the scikit-learn project or in any way affiliated with it. I'm just a very happy user.

What is Machine Learning?

```
# Import the example plot from the figures directory
from figures import plot_sgd_separator
plot_sgd_separator()
```



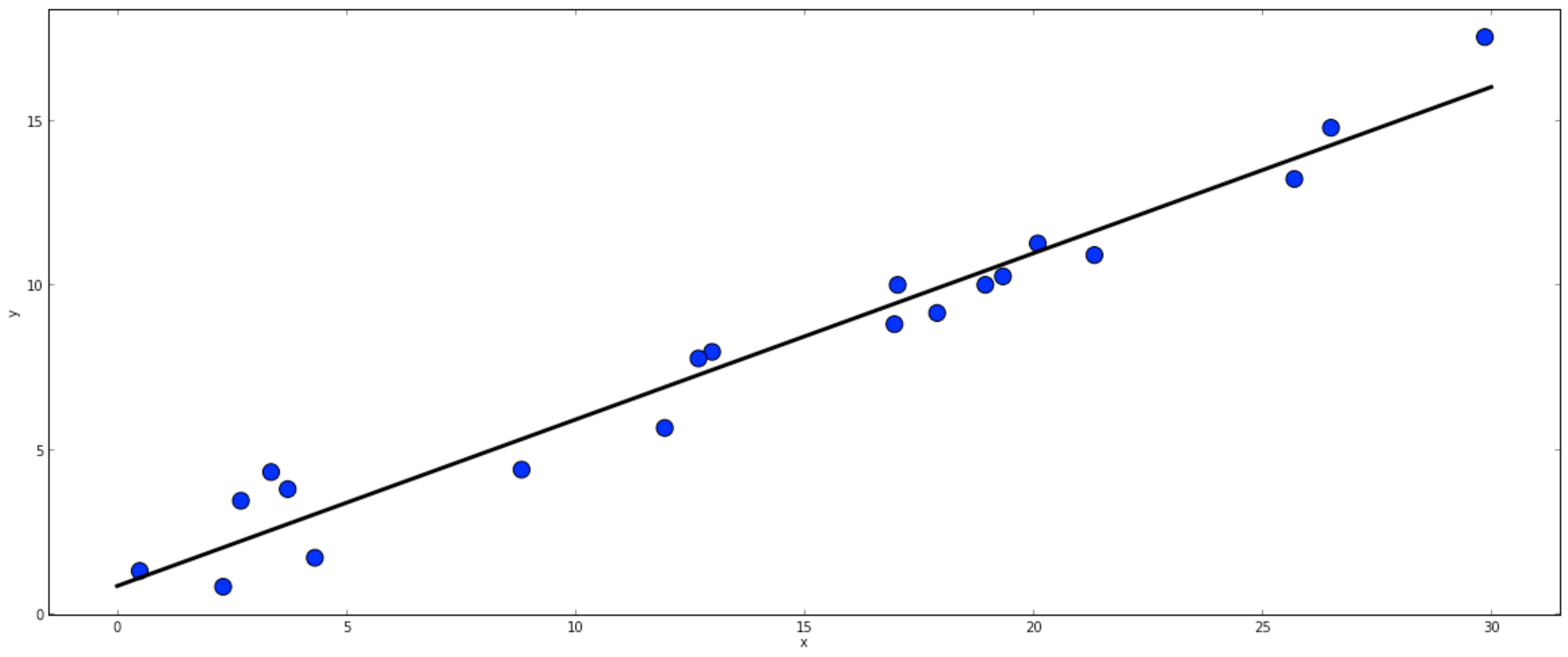
Machine learning algorithms can figure out how to perform important tasks based on previously seen data.

To illustrate this point, let's take a look at two simple machine learning tasks.

This plot represents data of two types. One is colored red; the other is colored blue. A classification algorithm may be used to draw a dividing line between the two clusters of points.

What is Machine Learning?

```python
from figures import plot_linear_regression
plot_linear_regression()
```



This plot shows a series of values that appear correlated. A plot like this could for example represent the prices of houses on the y axis and the square footage of those houses on the x axis.

We can pretty easily fit a line to this set of data.

Again, this is an example of fitting a model to data, such that the model can make generalizations about new data.

The model has been learned from the training data, and can be used to predict the result of test data: we might be given an x-value (square footage), and the model would allow us to predict the y value (price).

Again, this might seem like a trivial task, but it is a basic example of the type of problem you can solve with Machine Learning.

Data in scikit-learn

- Stored as a 2d-array
- [n_samples, n_features]
- n_samples: items to process
- n_features: distinct traits

Data in scikit-learn is usually represented as a 2d-array..
The size of the array is expected to be [n_samples by n_features]

n_samples refers to the number of samples: each sample is an item to process. A sample can be a document, a picture, a sound, a row in a database, or anything you can describe with a fixed set of quantitative traits.
n_features refers to the number of features or distinct traits that can be used to describe each item in a quantitative manner. Features are generally real-valued, but may be boolean or discrete-valued in some cases.
You have to choose your features in advance, but you can have as few or as many features as you want. Some of your features can represent traits that are relatively rare in your data set. In this case this feature would be set to zero for samples where it is not found.

The Iris Dataset

```
from IPython.core.display import Image, display

display(Image(filename='files/iris_setosa.jpg'))
print "Iris Setosa"
```

As an example of a simple dataset, let's look at the iris dataset which comes with scikit-learn.

The data consists of measurements of three different species of irises.

The Iris Dataset

```
display(Image(filename='files/iris_versicolor.jpg'))
print "Iris Versicolor"
```

The data contains (4) features for each sample. Each sample represents an individual flower.

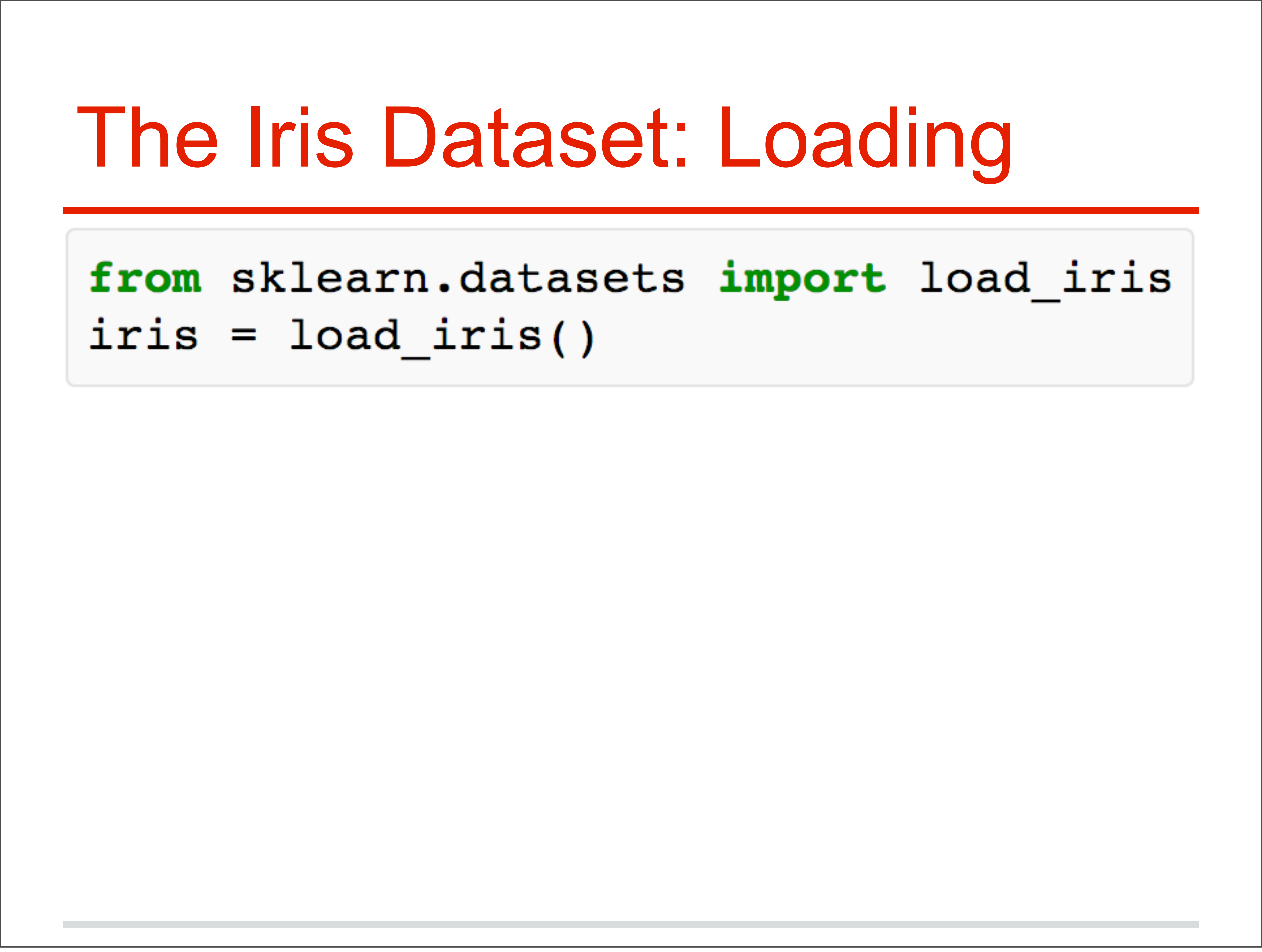
For each flower the features are:

- sepal length
- sepal width
- petal length
- petal width

The Iris Dataset

```
display(Image(filename='files/iris_virginica.jpg'))
print "Iris Virginica"
```

The iris dataset also contains the species of flower which is one of 3 classes.



The Iris Dataset: Loading

```
from sklearn.datasets import load_iris
iris = load_iris()
```

scikit-learn embeds a copy of the iris data along with a helper function to load it into numpy arrays

The Iris Dataset: Loading

```
from sklearn.datasets import load_iris
iris = load_iris()
```

```
iris.keys()
```

```
['target_names', 'data', 'target',
'DESCR', 'feature_names']
```

The resulting dataset is called a “Bunch” object: you can see what's available using the method keys()

The Iris Dataset: Loading

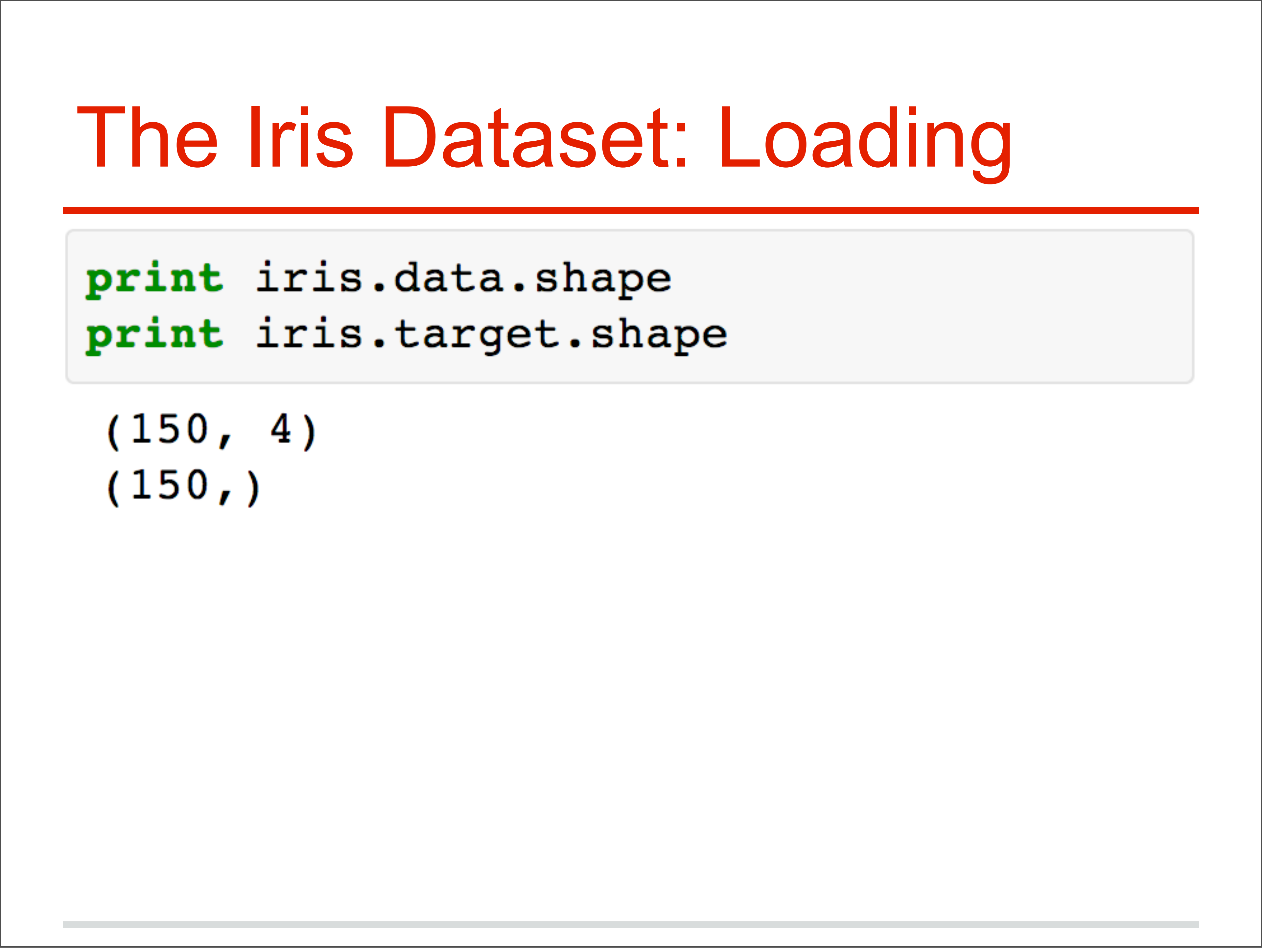
```
n_samples, n_features = iris.data.shape
print n_samples
print n_features
print iris.data[0]
```

```
150
4
[ 5.1  3.5  1.4  0.2]
```

The features of the sampled flowers are stored in the data attribute of the dataset

Data is a 2d array of 150 samples (by) 4 features

Here we can see what an individual sample looks like



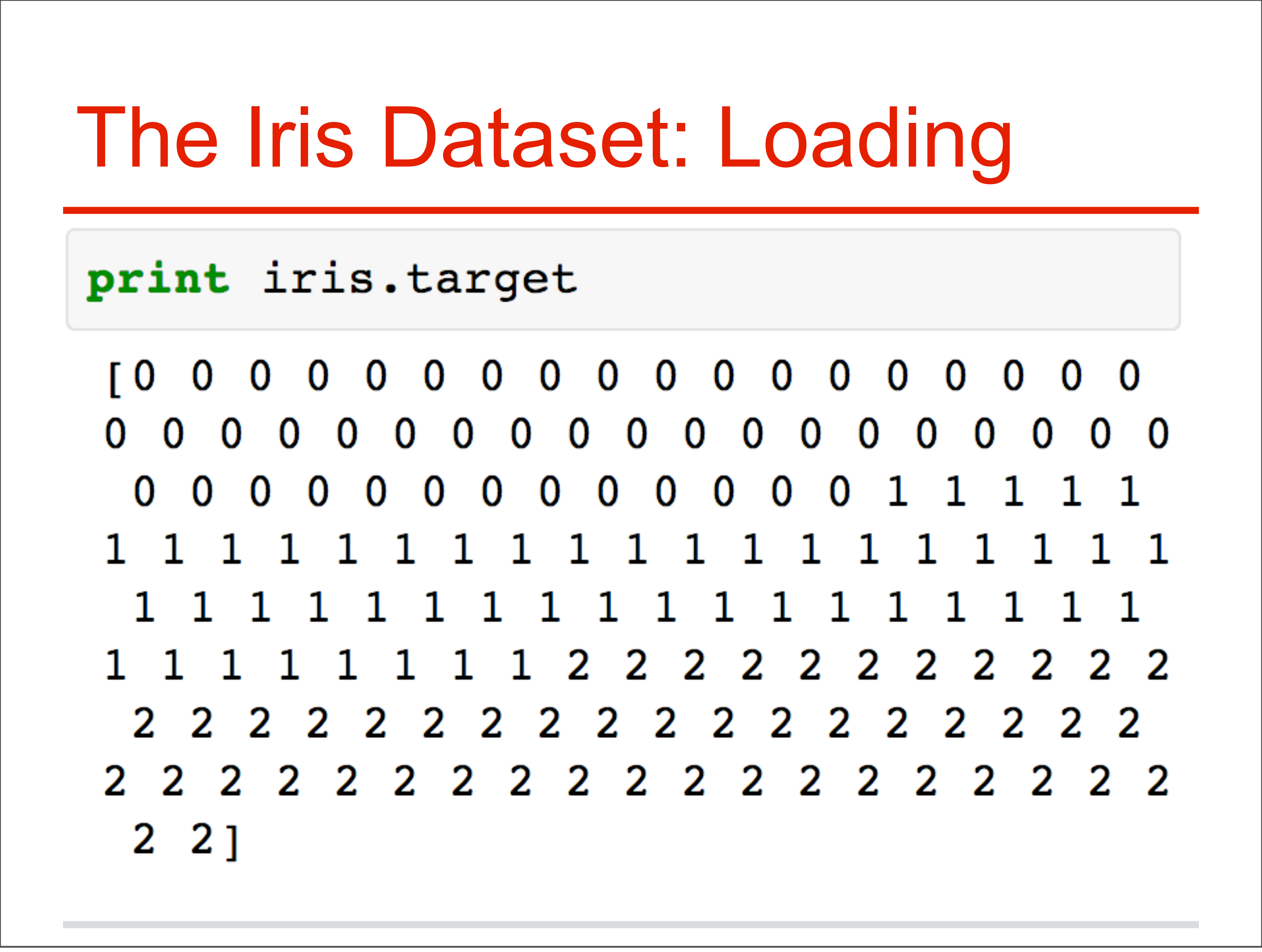
The Iris Dataset: Loading

```
print iris.data.shape
print iris.target.shape
```

```
(150, 4)
(150,)
```

The information about the class of each sample is stored in the target attribute of the dataset

While data is a 2d array...



The Iris Dataset: Loading

```
print iris.target
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

target is a 1d array with 1 class per sample (150).

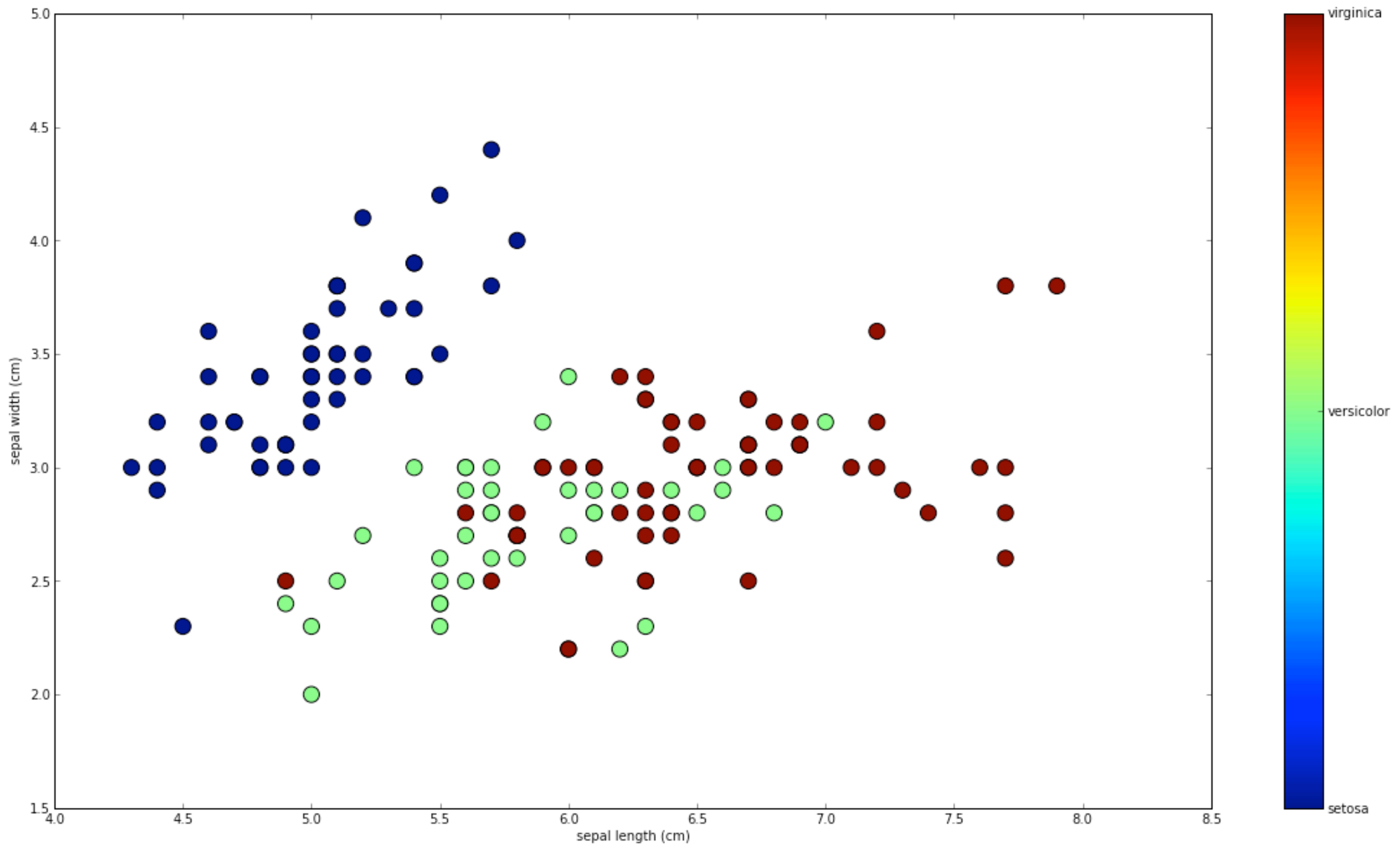
The Iris Dataset: Loading

```
print iris.target_names
```

```
['setosa' 'versicolor' 'virginica']
```

The names of the classes are stored in the target_names attribute. This can be used to convert the numerical target values to a human readable format.

The Iris Dataset: Loading

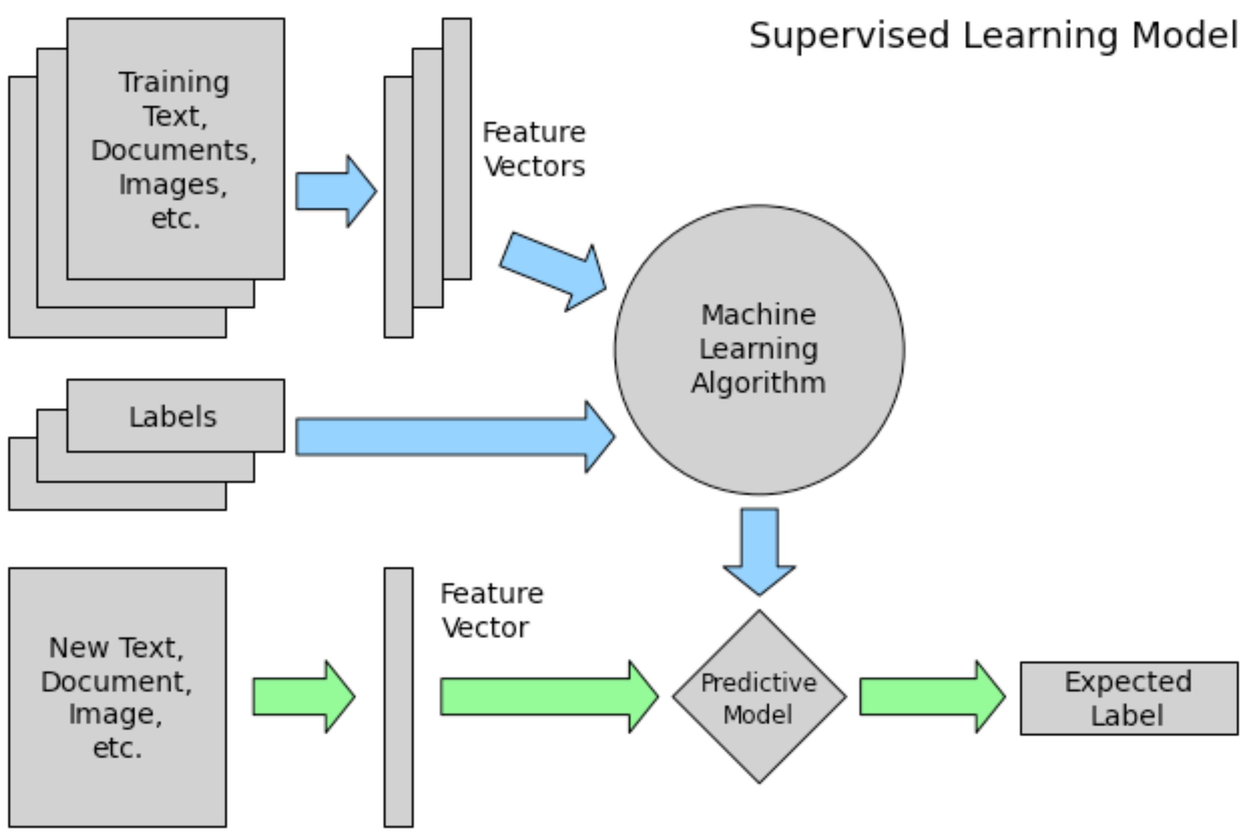


The iris data has 4 features. We can't easily visualize all 4 features plus the labels in a 2 or 3 dimensional graph. However one method for visualizing this data could be to plot two of the dimensions using a simple scatter-plot.

In this plot we've graphed:
y-axis: sepal width
x-axis: sepal length

The blue class seems reasonably distinct in this visualization. Unfortunately, it's hard to visually separate the green and the red classes using this technique.

Machine Learning: Supervised

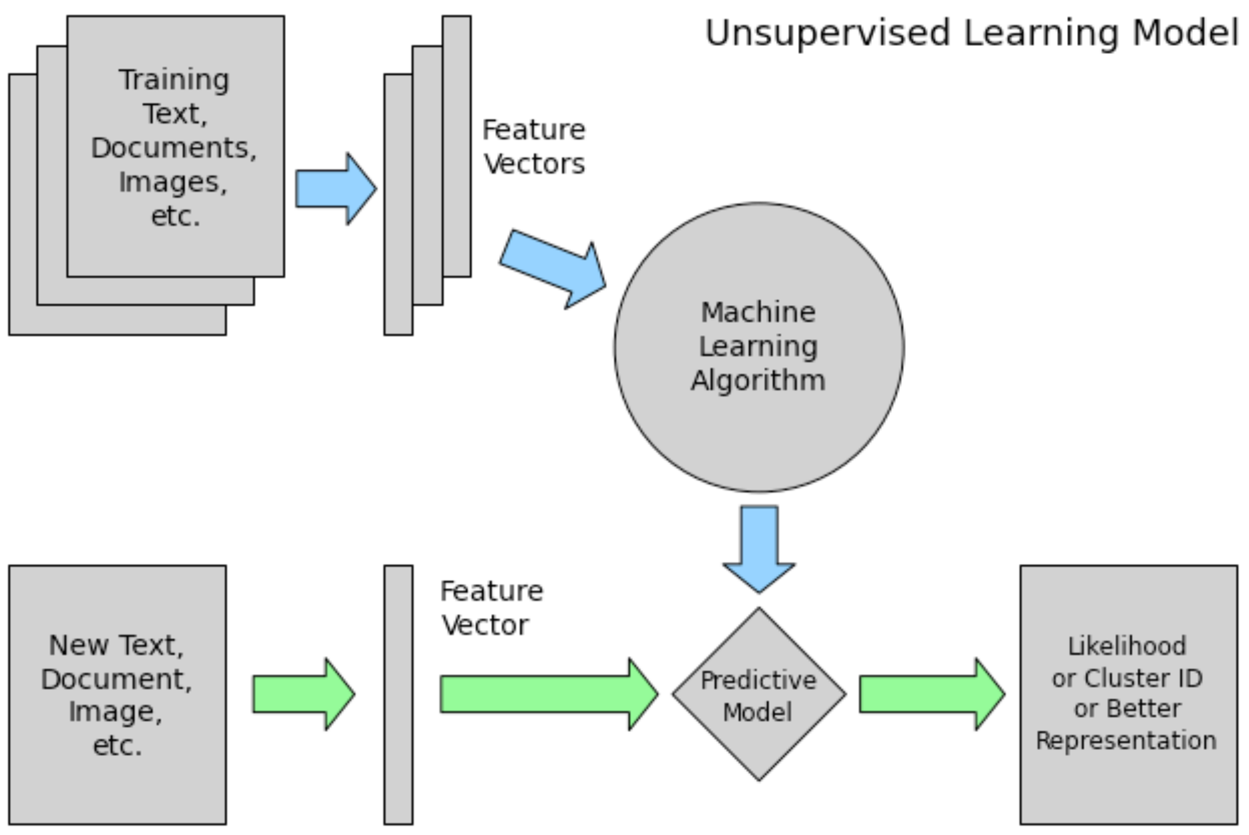

Supervised Learning Model
Training Text, Documents, Images, etc.
Feature Vectors
Machine Learning Algorithm
Labels
New Text, Document, Image, etc.
Feature Vector
Predictive Model
Expected Label

Now let’s explore the different types of machine learning.

Machine learning can be broken into two broad categories: supervised learning and unsupervised learning.

In Supervised Learning, we have a dataset consisting of both features and labels. The task is to construct an estimator which is able to predict the label of an object given the set of features. Using our iris data as an example, we could try to predict the species of iris given a set of measurements of its flower.

Machine Learning: Unsupervised


Unsupervised Learning Model
Training Text, Documents, Images, etc.
Feature Vectors
Machine Learning Algorithm
New Text, Document, Image, etc.
Feature Vector
Predictive Model
Likelihood or Cluster ID or Better Representation

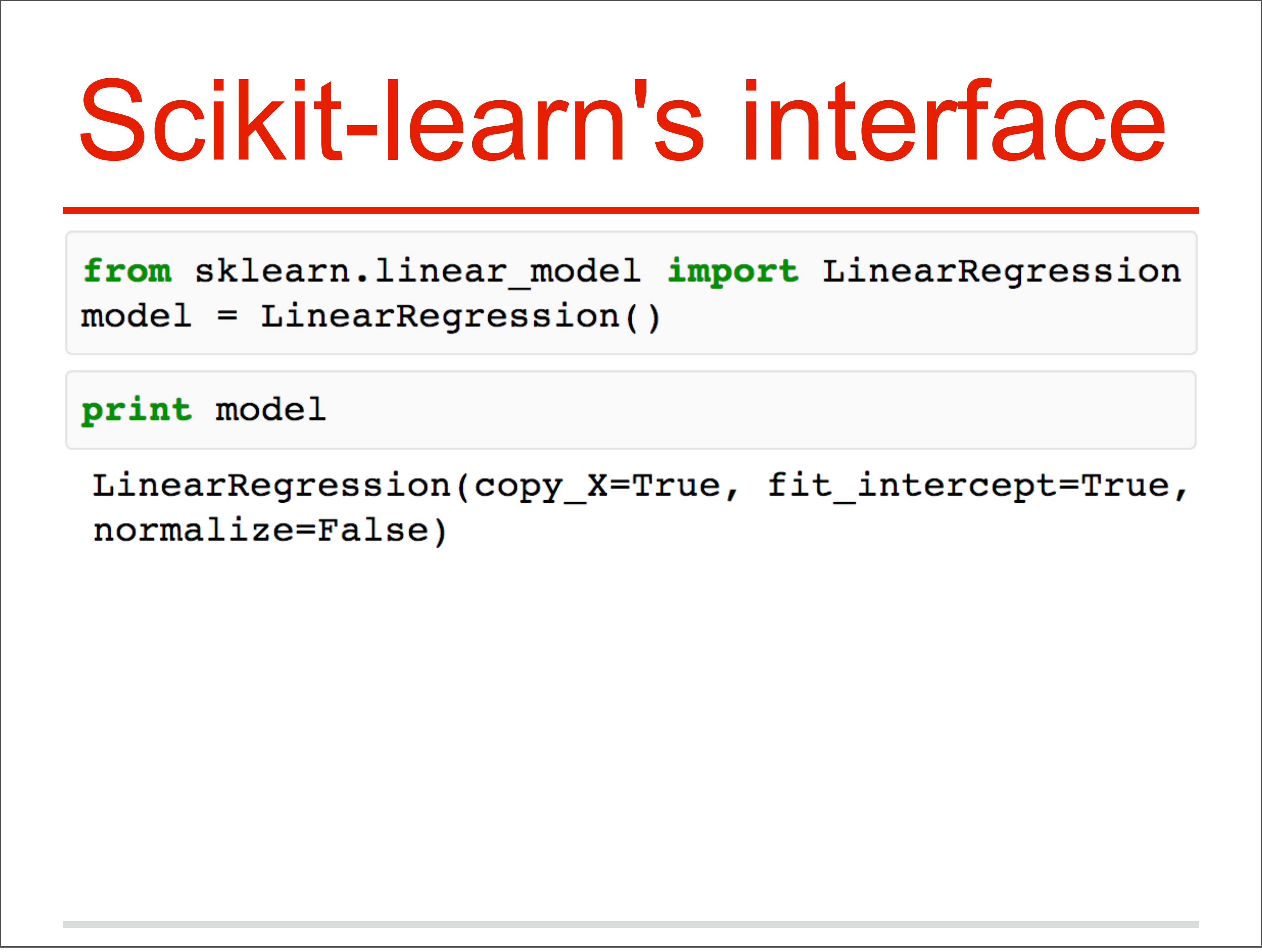
Unsupervised Learning addresses a different sort of problem. Here the data has no labels, and we are interested in finding similarities between the samples. You can think of unsupervised learning as a means of discovering labels from the data itself. Unsupervised learning comprises tasks such as dimensionality reduction and clustering. For example, in the iris data, we can used unsupervised methods to determine combinations of the features which are good at visualizing the structure of the data in 2 dimensions. We'll see an example of this later.

Sometimes you can even combine supervised and unsupervised learning. For example, unsupervised learning can be used to find useful features in the data, and then these features can be used within a supervised model.

Scikit-learn's interface

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

In scikit-learn, almost all operations are done through an estimator object.
For example, a linear regression estimator can be instantiated as follows:



Scikit-learn's interface

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

```
print model
```

```
LinearRegression(copy_X=True, fit_intercept=True,
normalize=False)
```

Scikit-learn strives to have a uniform interface across all methods. Given a scikit-learn estimator object (named model), the following methods are available:

All Estimators have a fit method. The fit method fits the model to a set of training data.

Supervised estimators can have a few methods.

- All supervised estimators have a predict method: given a trained model this method predicts the label of a new set of data.
- For classification problems, some estimators also provide the predict_proba method, which returns the probability that a new observation has each categorical label. In this case, the label with the highest probability is returned by the predict method.
- For classification or regression problems, most estimators implement a score method. Scores are between 0 and 1, with a larger score indicating a better fit. For most estimators score calculates the accuracy of the model.

Scikit-learn's interface

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

```
print model
```

```
LinearRegression(copy_X=True, fit_intercept=True,
normalize=False)
```

Unsupervised estimators have a few unique methods

The transform method transforms data into a new format. Some estimators implement the fit_transform method, which more efficiently performs a fit and a transform on the same input data.

Feature Extraction

Often data is unstructured & non-numerical:

- Text documents

often, data does not come in a nice, structured, CSV file where every column measures the same thing. Let's explore some common methods for extracting features in these cases.
Text documents:
Count the frequency of each word (n-grams) or pair of consecutive words in each document. This approach is called **Bag of Words**

Feature Extraction

Often data is unstructured & non-numerical:

- Text documents
- Images

Extracting features from Images:
Rescale the picture to a fixed size and take all the raw pixels values (with or without luminosity normalization)
Take some transformation of the image
Perform local feature extraction: split the image into small regions and perform feature extraction locally in each area, Then combine all the features of the individual areas into a single array.

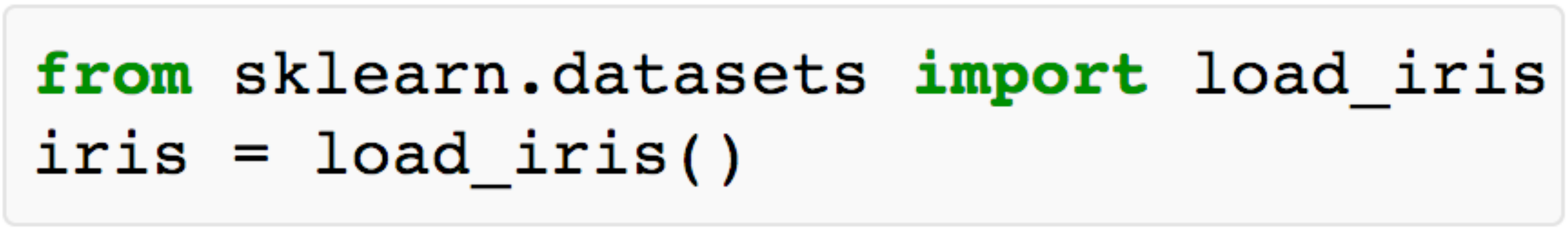
Feature Extraction

Often data is unstructured & non-numerical:

- Text documents
- Images
- Sounds

Extracting features from Sounds:
Same type of strategies as for images; the difference is it’s a 1D rather than 2D space.

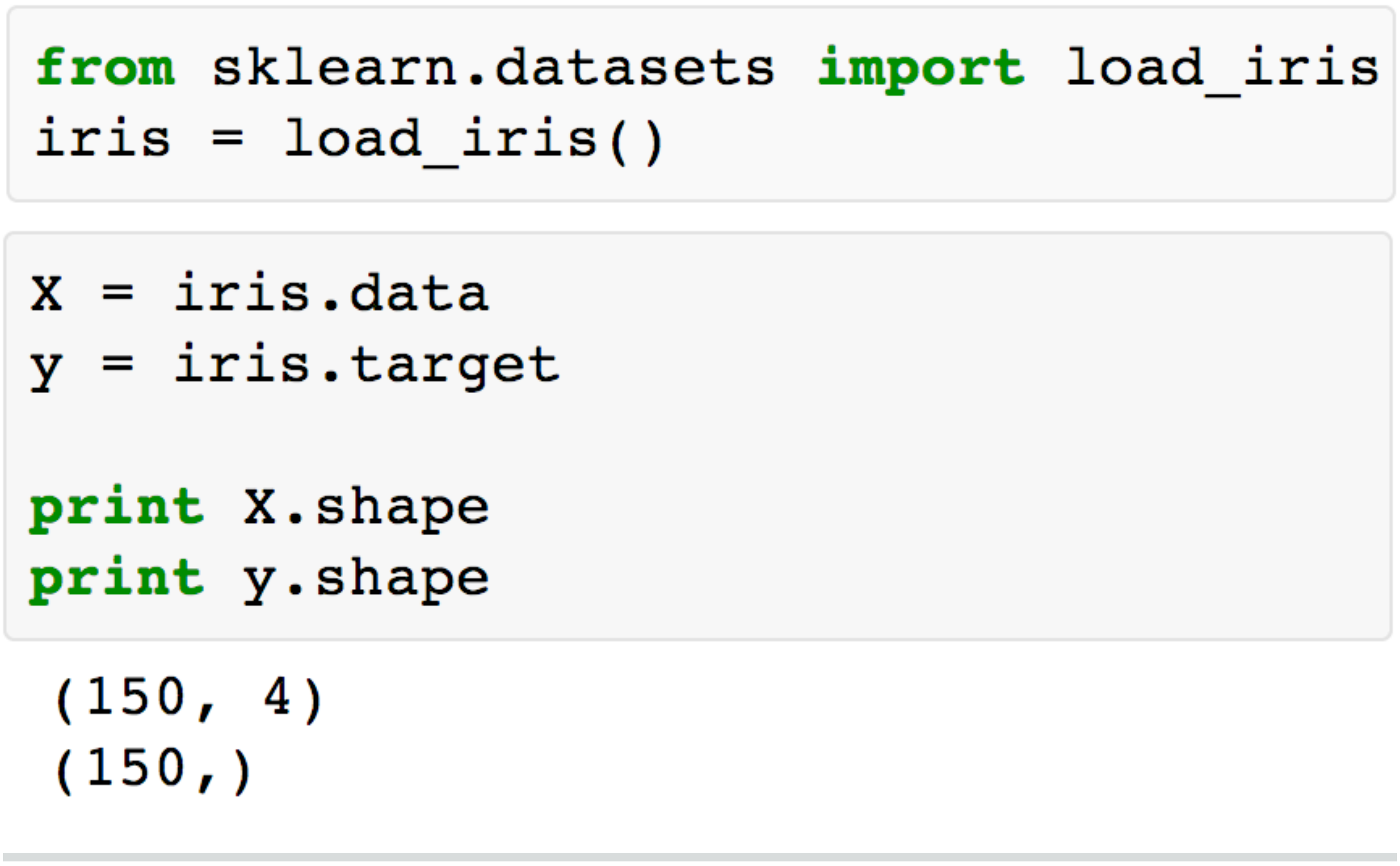
Supervised Learning: Classification


```python
from sklearn.datasets import load_iris
iris = load_iris()
```

Now that we’ve covered all the basics, let’s train a classification model using the iris dataset.

First lets load the iris data like before.

Supervised Learning: Classification



```
from sklearn.datasets import load_iris
iris = load_iris()
```

```
X = iris.data
y = iris.target

print X.shape
print y.shape
```

```
(150, 4)
(150,)
```

Let’s say that we were assigned the task of guessing the class of an individual flower given the measurements of petals and sepals. This is a classification task

You’ll note that we’re using the variable uppercase X to represent our data and the variable lower case y to present our targets. These two variables are frequently used in the Machine Learning field so you will likely see this format frequently.

Once the data has this format it is trivial to train a classifier...

Supervised Learning: Classification

```
from sklearn.svm import LinearSVC
```

...for example let's try out a support vector machine.

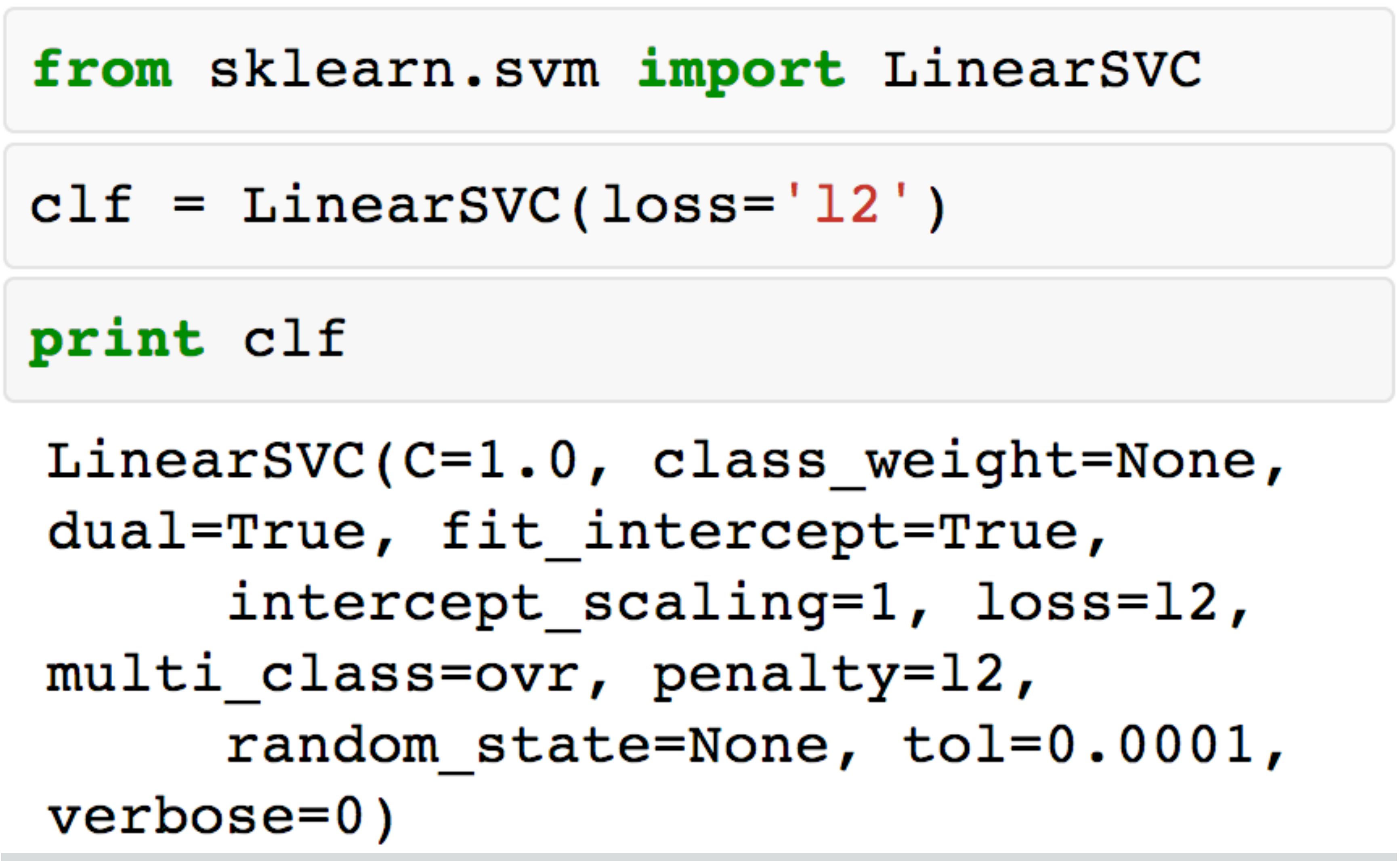
Supervised Learning: Classification

```python
from sklearn.svm import LinearSVC
```

```python
clf = LinearSVC(loss='l2')
```

The first thing to do is to create an instance of the classifier. This can be done simply by calling the class name, with any arguments that the object accepts

Supervised Learning: Classification



```
from sklearn.svm import LinearSVC
```

```
clf = LinearSVC(loss='l2')
```

```
print clf
```

```
LinearSVC(C=1.0, class_weight=None,
dual=True, fit_intercept=True,
     intercept_scaling=1, loss=l2,
multi_class=ovr, penalty=l2,
     random_state=None, tol=0.0001,
verbose=0)
```

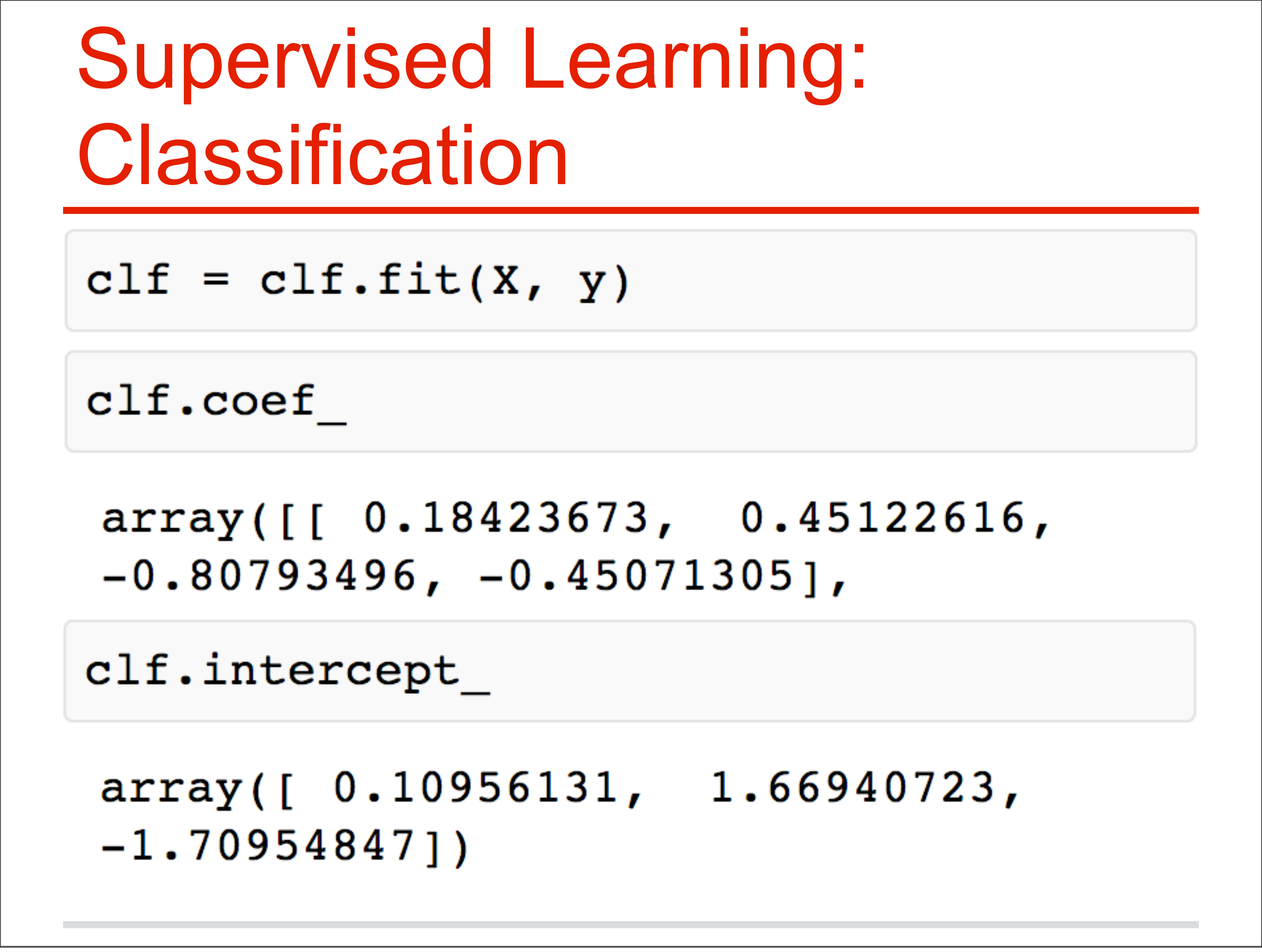
clf is a statistical model that has parameters that control the learning algorithm. Those parameters can be supplied by the user in the constructor of the model.

Each estimator has different parameters. There are several methods for choosing good values for each parameter. I won’t cover these methods in this talk, but these are covered in Jake Vanderplas’ ipython notebooks.

Supervised Learning: Classification

```
clf = clf.fit(X, y)
```

By default the model's fit parameters are not initialized. They will be tuned automatically from the data by calling the fit method with the data - X and labels - y



Supervised Learning: Classification

```
clf = clf.fit(X, y)
```

```
clf.coef_
```

```
array([[ 0.18423673,  0.45122616,
-0.80793496, -0.45071305],
```

```
clf.intercept_
```

```
array([ 0.10956131,  1.66940723,
-1.70954847])
```

We can now see some of the fit parameters within the classifier object.
In scikit-learn, parameters defined during training have a trailing underscore.

Supervised Learning: Classification

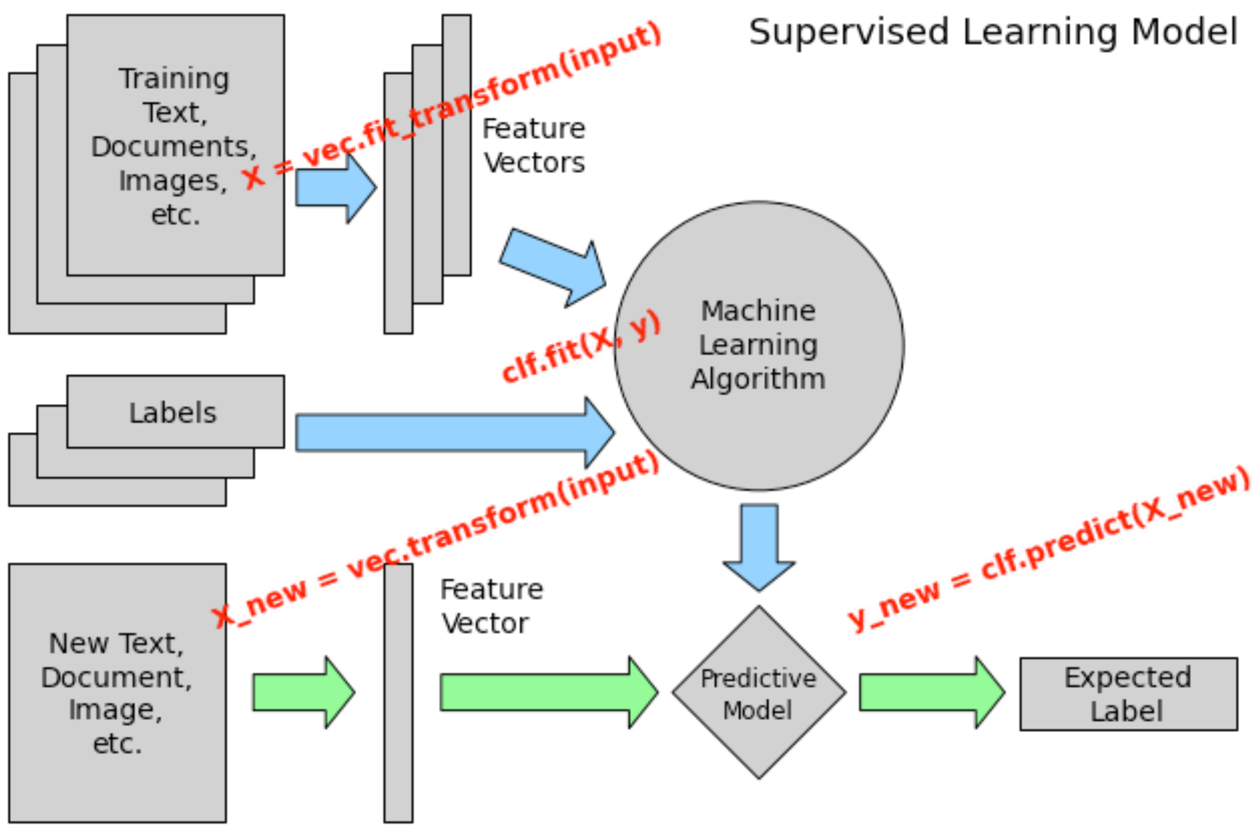
```
X_new = [[ 5.0,  3.6,  1.3,  0.25]]

clf.predict(X_new)
```

```
array([0])
```

Once the model is trained, it can be used to predict the most likely outcome on new data. For instance let us define a list of simple samples that looks like the first sample of the iris dataset:

Our model predicts the sample is of class 0

Supervised Learning: Classification


Supervised Learning Model
Training Text, Documents, Images, etc.
X = vec.fit_transform(input)
Feature Vectors
Labels
clf.fit(X, y)
Machine Learning Algorithm
New Text, Document, Image, etc.
X_new = vec.transform(input)
Feature Vector
Predictive Model
y_new = clf.predict(X_new)
Expected Label

So now that we’ve trained our first model, let’s revisit the previous diagram and see where our fit and predict calls fit in.

We can see that we called fit with our vectorized features. We were able to skip this step in this example because the iris dataset is already vectorized.

We can see that once fit was called, we called predict on a new data point and got as output an expected label.

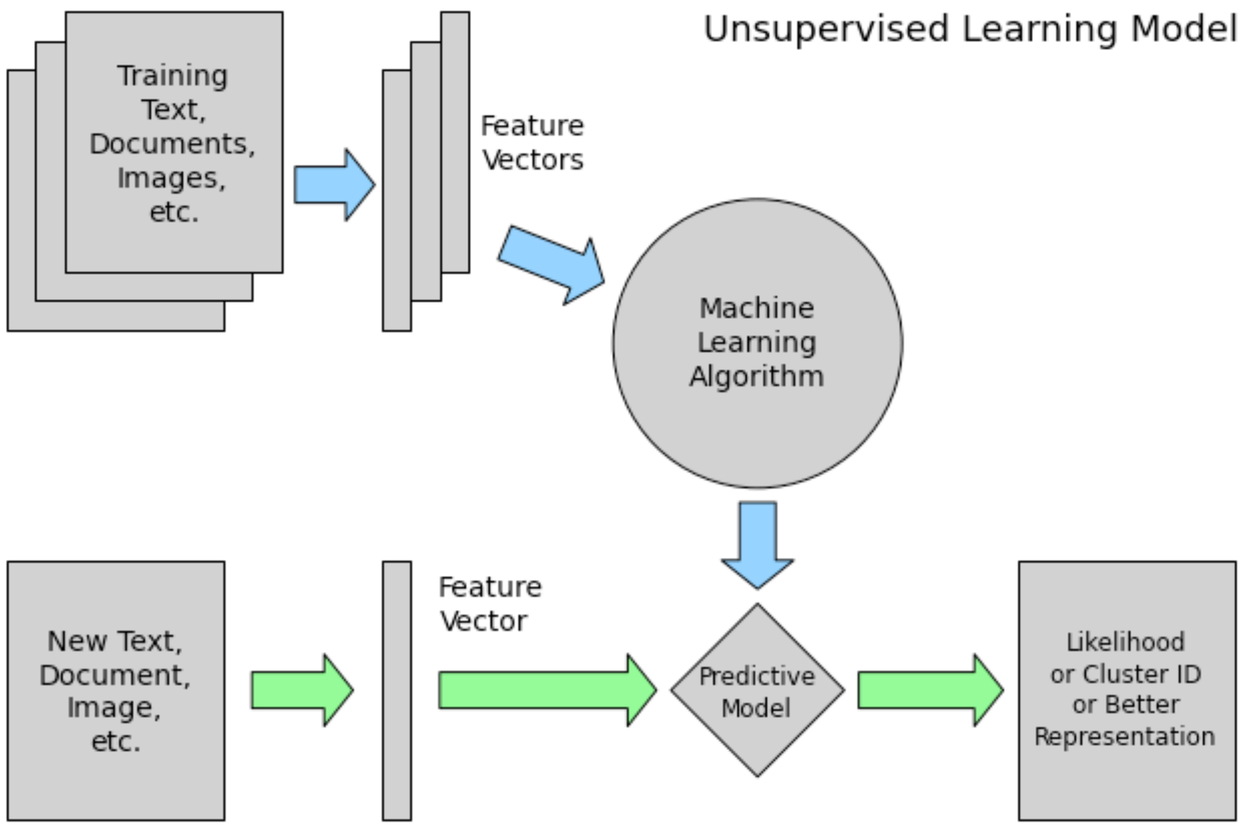
Supervised Learning: Classification

- Email classification
- Language identification
- New article categorization
- Sentiment analysis
- Facial recognition
- ...

Classification involves predicting an unknown category based on observed features.
Let’s go over a few examples of interesting classification tasks:

E-mail classification: labeling email as spam or ham
Language identification: labeling documents as English, Spanish, etc...
News articles categorization: labeling articles as business, technology...
Sentiment analysis: labeling customer feedback as negative, neutral, positive
Facial recognition: label images as matching or not matching a person

Unsupervised Learning


Unsupervised Learning Model
Training Text, Documents, Images, etc.
Feature Vectors
Machine Learning Algorithm
New Text, Document, Image, etc.
Feature Vector
Predictive Model
Likelihood or Cluster ID or Better Representation

Let's revisit unsupervised learning. The major difference between supervised and unsupervised learning is that in the case of unsupervised learning, our data is unlabeled.

Previously we visualized the iris data by plotting pairs of dimensions.

Here we will use an unsupervised dimensionality reduction algorithm to improve on our previous technique.

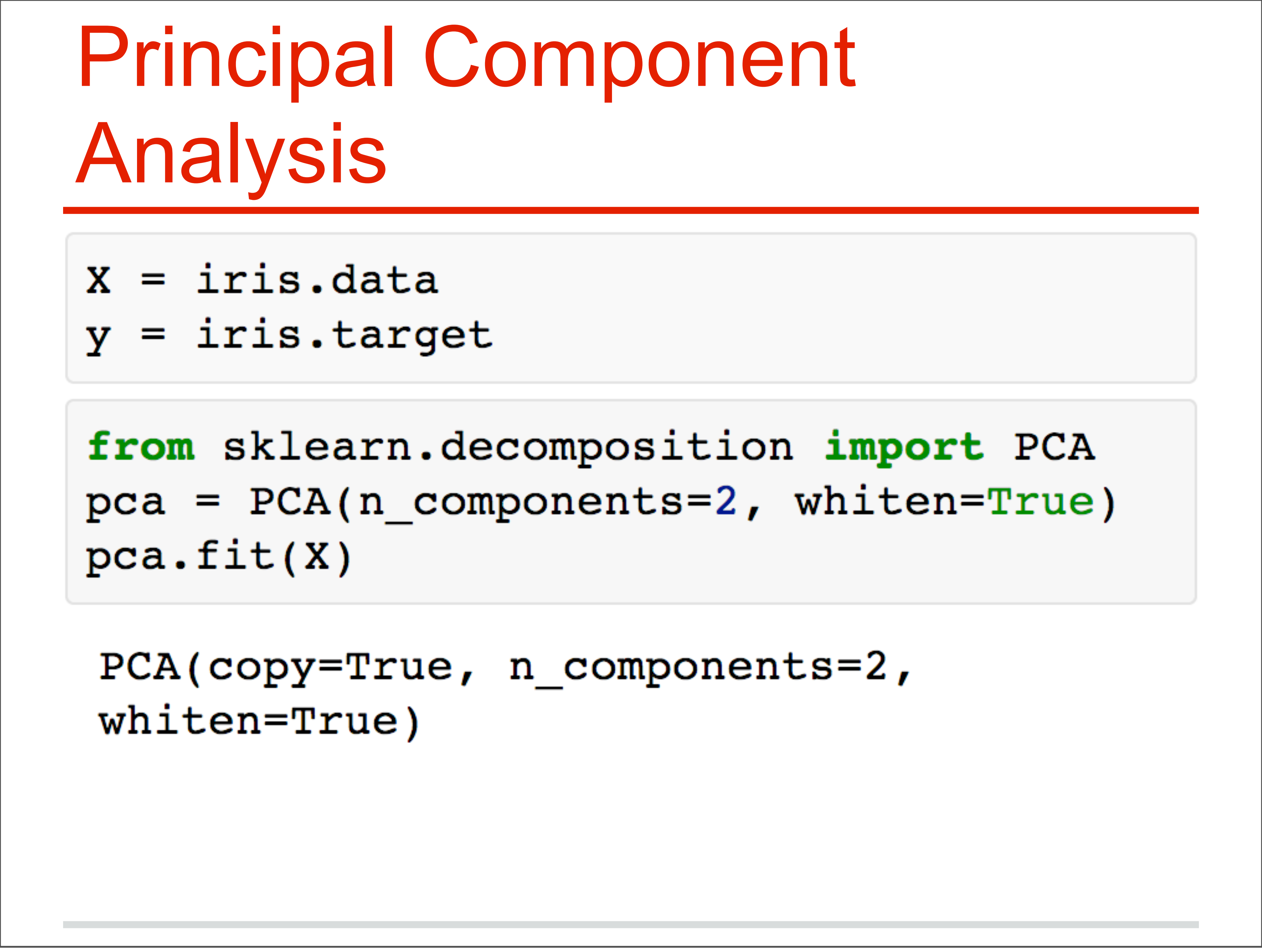
Dimensionality Reduction

```
X = iris.data
y = iris.target
```

Dimensionality reduction is the task of deriving a set of new abstract features that is smaller than the original feature set while retaining most of the variance of the original data.

Here we'll use a common but powerful dimensionality reduction technique called Principal Component Analysis (PCA). We'll perform PCA on the iris dataset that we saw before:

Since this is unsupervised learning, target (y) will be unused, however we'll use it later to visualize our results.



Principal Component Analysis

```
X = iris.data
y = iris.target
```

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2, whiten=True)
pca.fit(X)
```

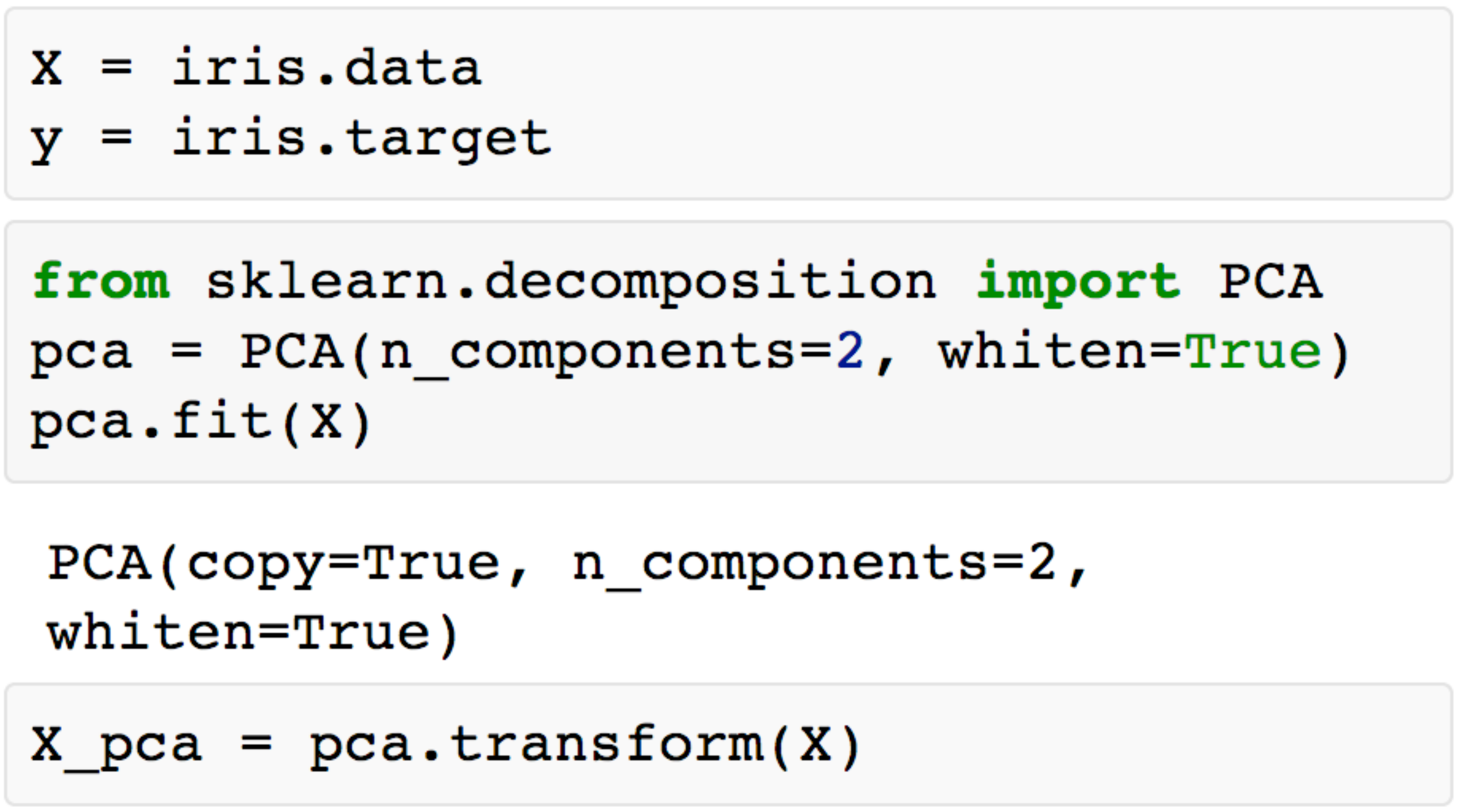
```
PCA(copy=True, n_components=2,
whiten=True)
```

PCA allows you to re-express a set of data points in terms of basic components that explain the most variance in the data.

This is accomplished by combining the original features.

If the number of retained components is 2 or 3, PCA can be used to visualize the dataset.

Unsupervised Learning: PCA



```
X = iris.data
y = iris.target
```

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2, whiten=True)
pca.fit(X)
```

```
PCA(copy=True, n_components=2,
whiten=True)
```

```
X_pca = pca.transform(X)
```

We’ve used PCA to transform our original 4d data into 2d data

Unsupervised Learning: PCA

```
import numpy as np
np.round(X_pca.mean(axis=0), decimals=5)
```

```
array([-0.,  0.])
```

PCA normalizes and whitens the data, which means that the data is now centered on both components

The mean of both of the artificial components is 0...

Unsupervised Learning: PCA

```python
import numpy as np
np.round(X_pca.mean(axis=0), decimals=5)
```

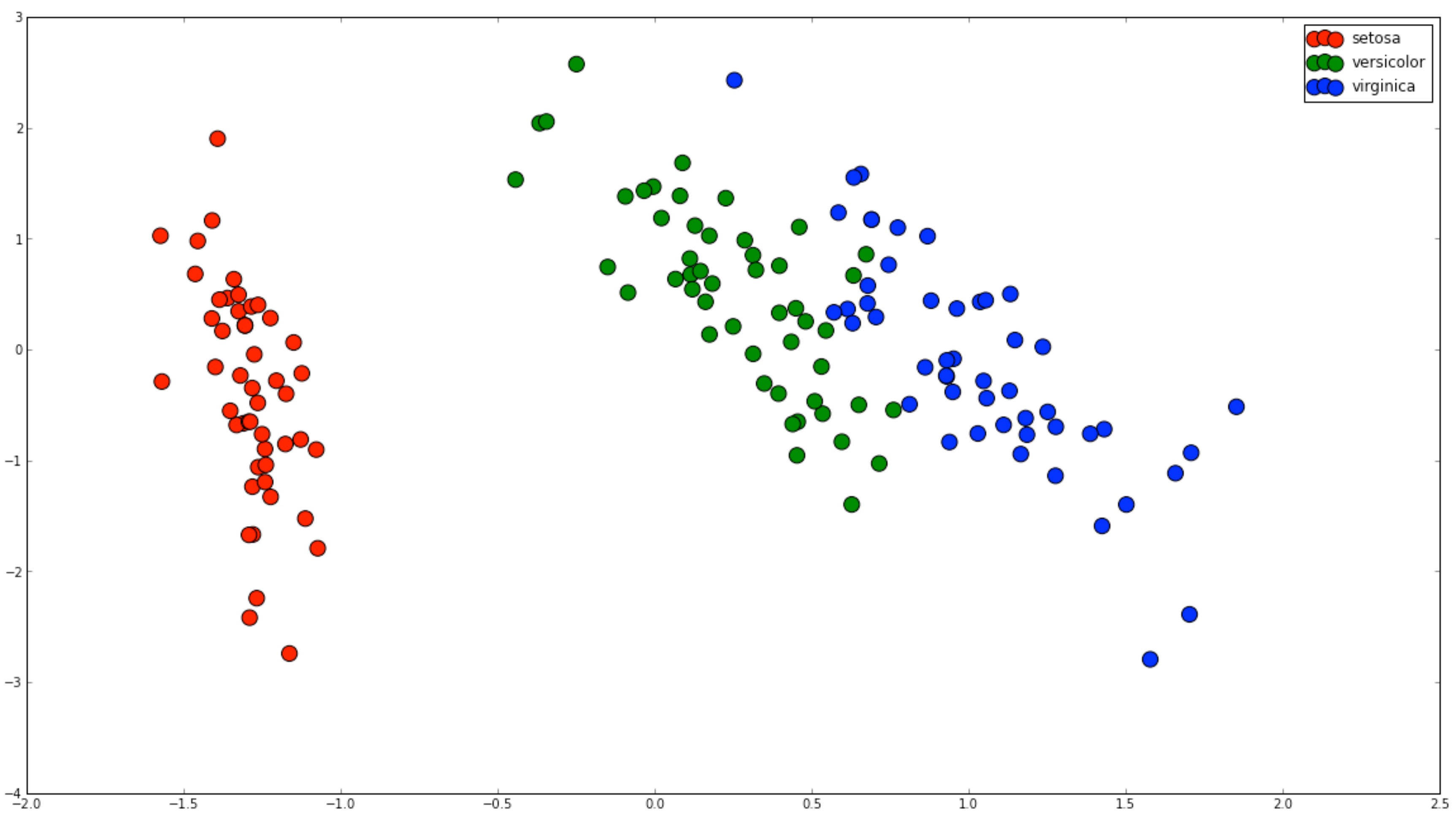
```
array([-0.,  0.])
```

```python
np.round(X_pca.std(axis=0), decimals=5)
```

```
array([ 1.,  1.])
```

... and the standard deviation is 1

Unsupervised Learning: PCA



Now we can visualize the iris dataset along the two new dimensions

Note that this visualization was generated without any information about the labels (y) (represented by the colors): this is the sense in which the learning is unsupervised.

Even so, we see that the projection gives us insight into the distribution of the different flowers: notably, the red class is much more distinct than the other two species.

And even among the green and blue classes, there is a pretty good division line that can be drawn.

Validation & Testing

```
# Get the data
X = iris.data
y = iris.target
```

The last thing we’ll cover in this talk is validation and testing.

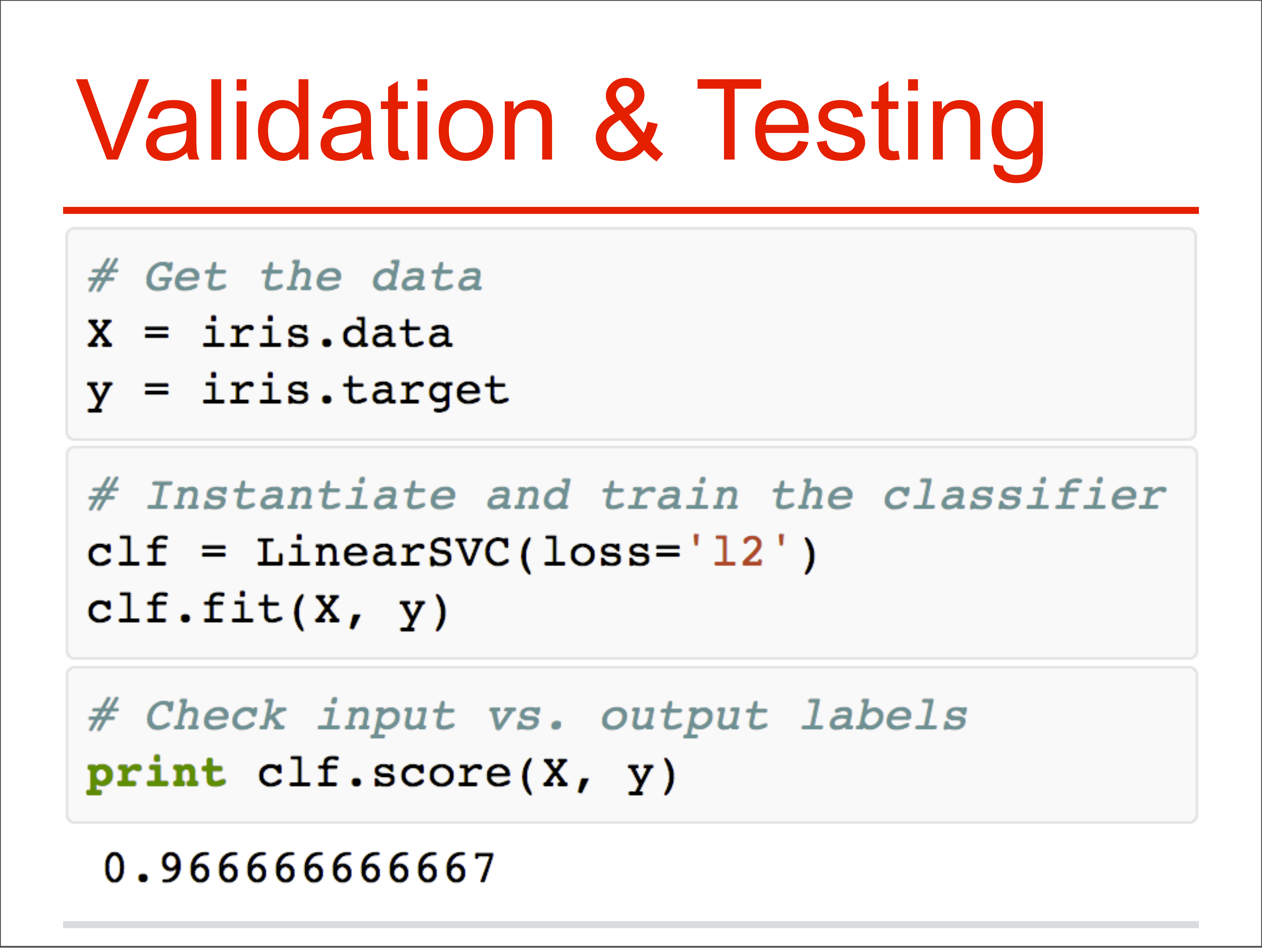
The most common mistake beginners make when training statistical models is to evaluate the quality of the model on the same data used for fitting the model.

Validation & Testing

```
# Get the data
X = iris.data
y = iris.target
```

```
# Instantiate and train the classifier
clf = LinearSVC(loss='l2')
clf.fit(X, y)
```

Here we're training the classifier with all the data.



Validation & Testing

```
# Get the data
X = iris.data
y = iris.target
```

```
# Instantiate and train the classifier
clf = LinearSVC(loss='l2')
clf.fit(X, y)
```

```
# Check input vs. output labels
print clf.score(X, y)
```

```
0.966666666667
```

We’re getting pretty high accuracy with this model.

Question: what might be the problem with this approach?

Overfitting

The problem is that some models can be subject to overfitting: they can learn the training data by heart without generalizing. The symptoms are:

- The accuracy on the data used for training can be excellent (sometimes 100%)
- The models do little better than random predictions when facing new data that was not part of the training set

If you evaluate your model on your training data you won’t be able to tell whether your model is overfitting or not.

Cross-Validation

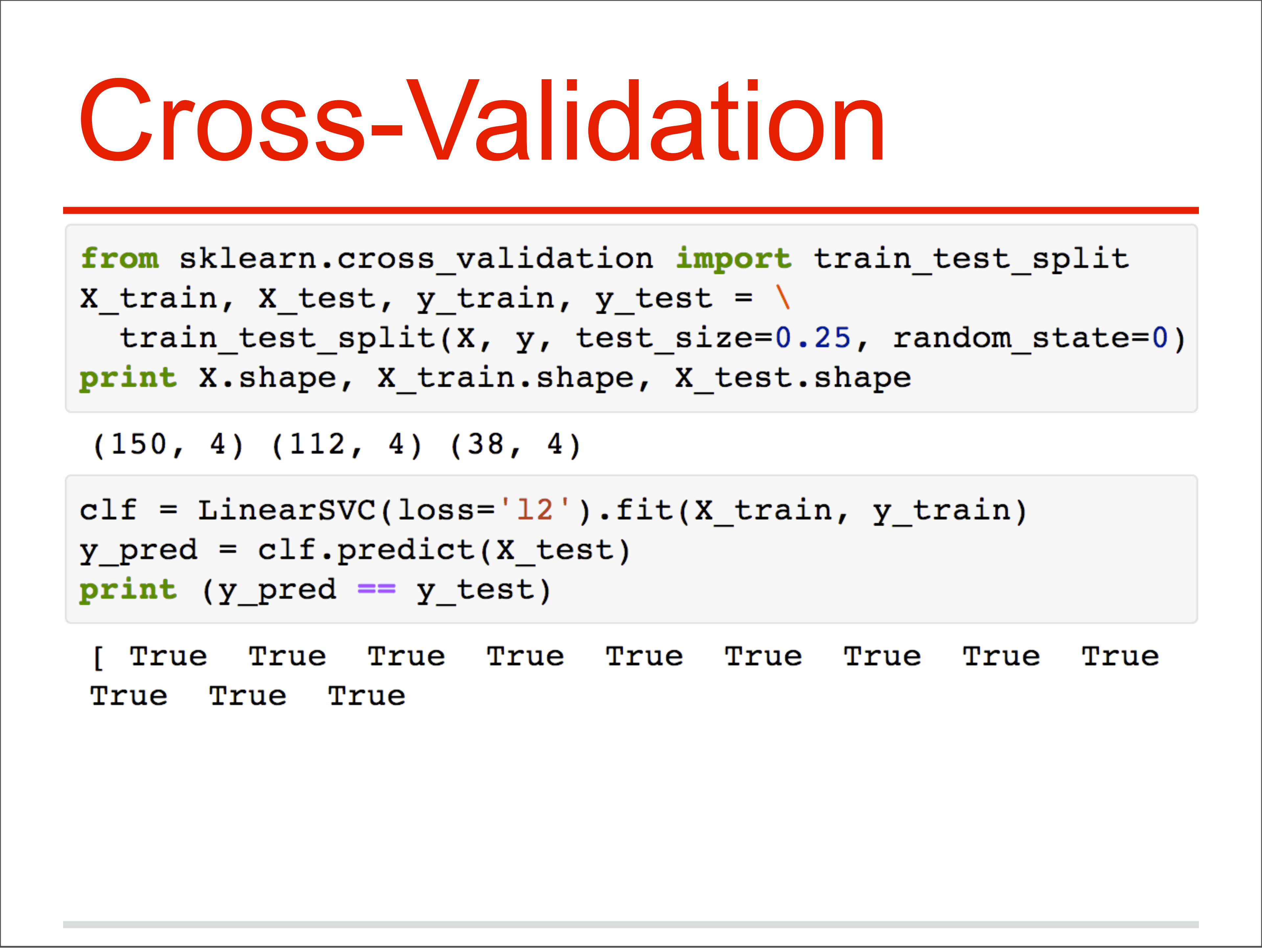
```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = \
  train_test_split(X, y, test_size=0.25, random_state=0)
print X.shape, X_train.shape, X_test.shape
```

```
(150, 4) (112, 4) (38, 4)
```

Learning the parameters of a prediction function and testing it on the same data is a mistake: a model that would just repeat the labels of the samples that it has seen would have a perfect score but would fail to predict anything useful on new data.
To avoid over-fitting, we have to define two different sets:

- a training set X_train, y_train which is used for training the model
- a testing set X_test, y_test which is used for evaluating the fitted model

In scikit-learn such a random split can be quickly computed with the train_test_split helper function.



Cross-Validation

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = \
  train_test_split(X, y, test_size=0.25, random_state=0)
print X.shape, X_train.shape, X_test.shape
```

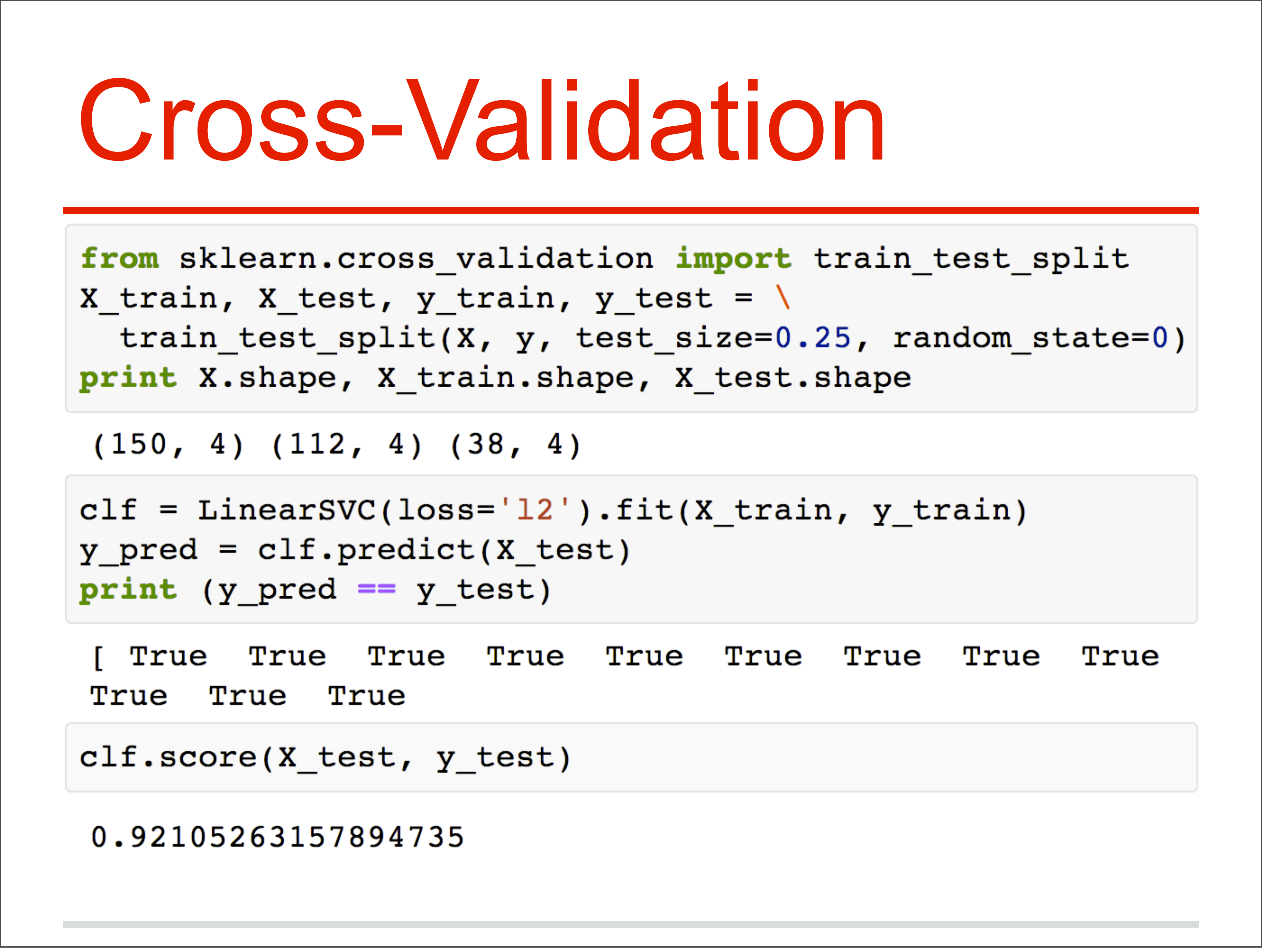
```
(150, 4) (112, 4) (38, 4)
```

```
clf = LinearSVC(loss='l2').fit(X_train, y_train)
y_pred = clf.predict(X_test)
print (y_pred == y_test)
```

```
[ True  True  True  True  True  True  True  True  True
True  True  True
```

using train_test_split, we can train on the training data...

...and test on the testing data



Cross-Validation

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = \
  train_test_split(X, y, test_size=0.25, random_state=0)
print X.shape, X_train.shape, X_test.shape
```

```
(150, 4) (112, 4) (38, 4)
```

```
clf = LinearSVC(loss='l2').fit(X_train, y_train)
y_pred = clf.predict(X_test)
print (y_pred == y_test)
```

```
[ True  True  True  True  True  True  True  True  True
True  True  True
```

```
clf.score(X_test, y_test)
```

```
0.9210526315789473
```

There is an issue here, however:

by defining these two sets, we significantly reduce the number of samples which can be used for training the model, and the results can depend on a particular random choice for the pair of (train, test) sets.

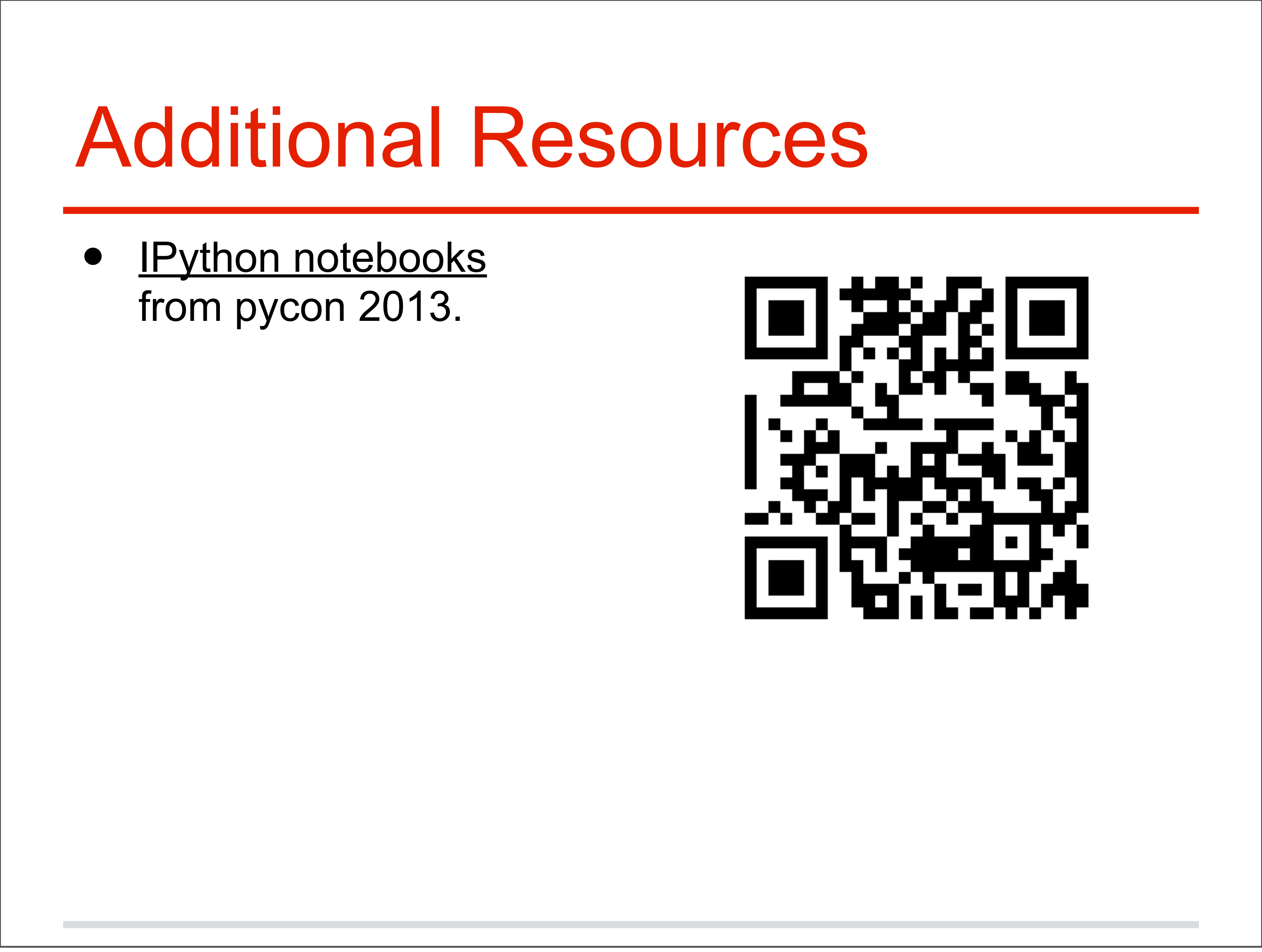
A solution is to split the whole dataset a few times randomly into different training and testing sets, and to calculate the average value of the prediction scores obtained with the different sets.

Such a procedure is called cross-validation. This approach can be computationally expensive, but does not waste too much data.

Information on cross validation, and a lot of other awesome things which I haven't covered can be found in the following resources.

Additional Resources

- Machine Learning 101 tutorial from scikit-learn.



Additional Resources

- IPython notebooks
  from pycon 2013.

Thanks go to Jake Vanderplas for creating an excellent set of ipython notebooks for pycon 2013 which I've used for my code samples.

My info

Tweet me @beckerfuffle
Find me at beckerfuffle.com

These slides and more @ github.com/mdbecker

You can find me online @beckerfuffle on Twitter. At beckerfuffle.com, and I'm also mdbecker on github. I'll be posting the materials for this talk on my github.