

# Visualizing the PCA transformation

UNSUPERVISED LEARNING IN PYTHON



**Benjamin Wilson**

Director of Research at lateral.io

# Dimension reduction

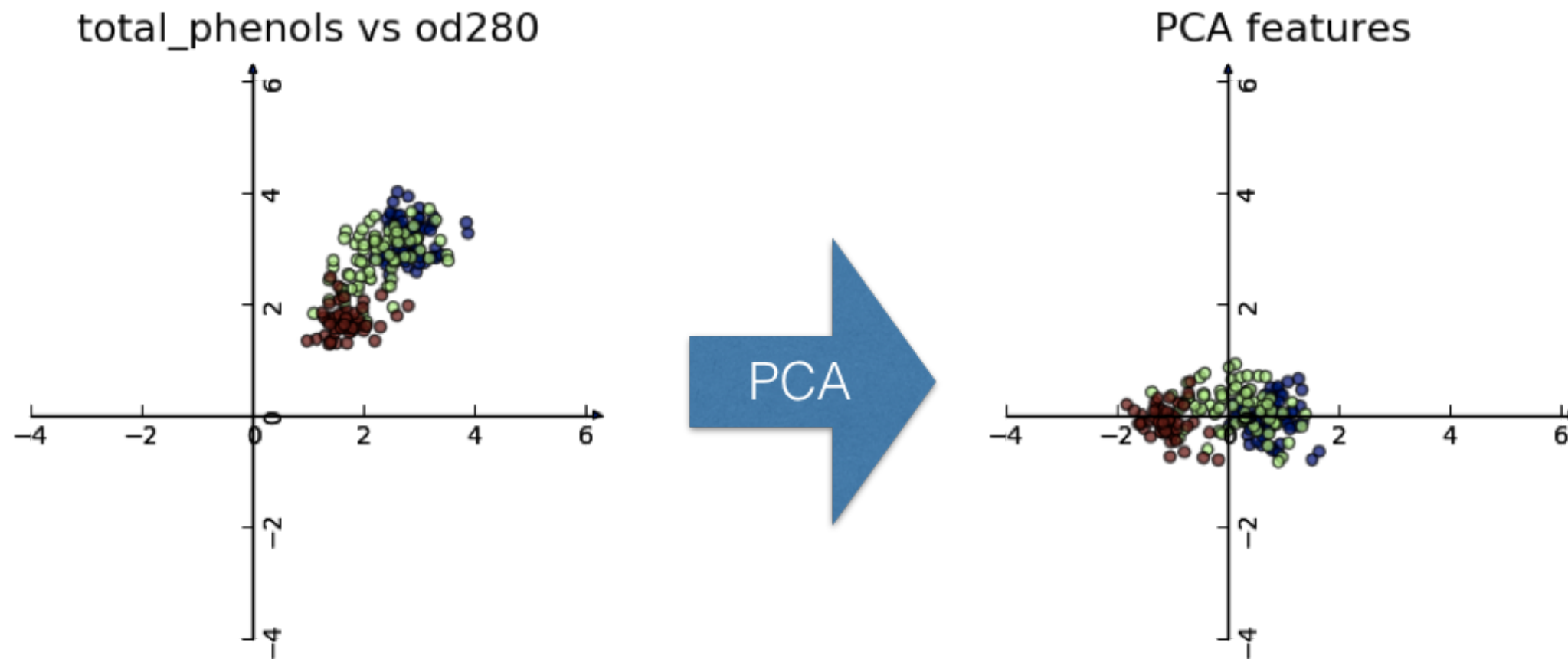
- More efficient storage and computation
- Remove less-informative "noise" features
- ... which cause problems for prediction tasks, e.g. classification, regression

# Principal Component Analysis

- PCA = "Principal Component Analysis"
- Fundamental dimension reduction technique
- First step "decorrelation" (considered here)
- Second step reduces dimension (considered later)

# PCA aligns data with axes

- Rotates data samples to be aligned with axes
- Shifts data samples so they have mean 0
- No information is lost



# PCA follows the fit/transform pattern

- `PCA` is a scikit-learn component like `KMeans` or `StandardScaler`
- `fit()` learns the transformation from given data
- `transform()` applies the learned transformation
- `transform()` can also be applied to new data

# Using scikit-learn PCA

- `samples` = array of two features ( `total_phenols` & `od280` )

```
[[ 2.8   3.92]  
...  
[ 2.05  1.6 ]]
```

```
from sklearn.decomposition import PCA  
model = PCA()  
model.fit(samples)
```

```
PCA()
```

```
transformed = model.transform(samples)
```

# PCA features

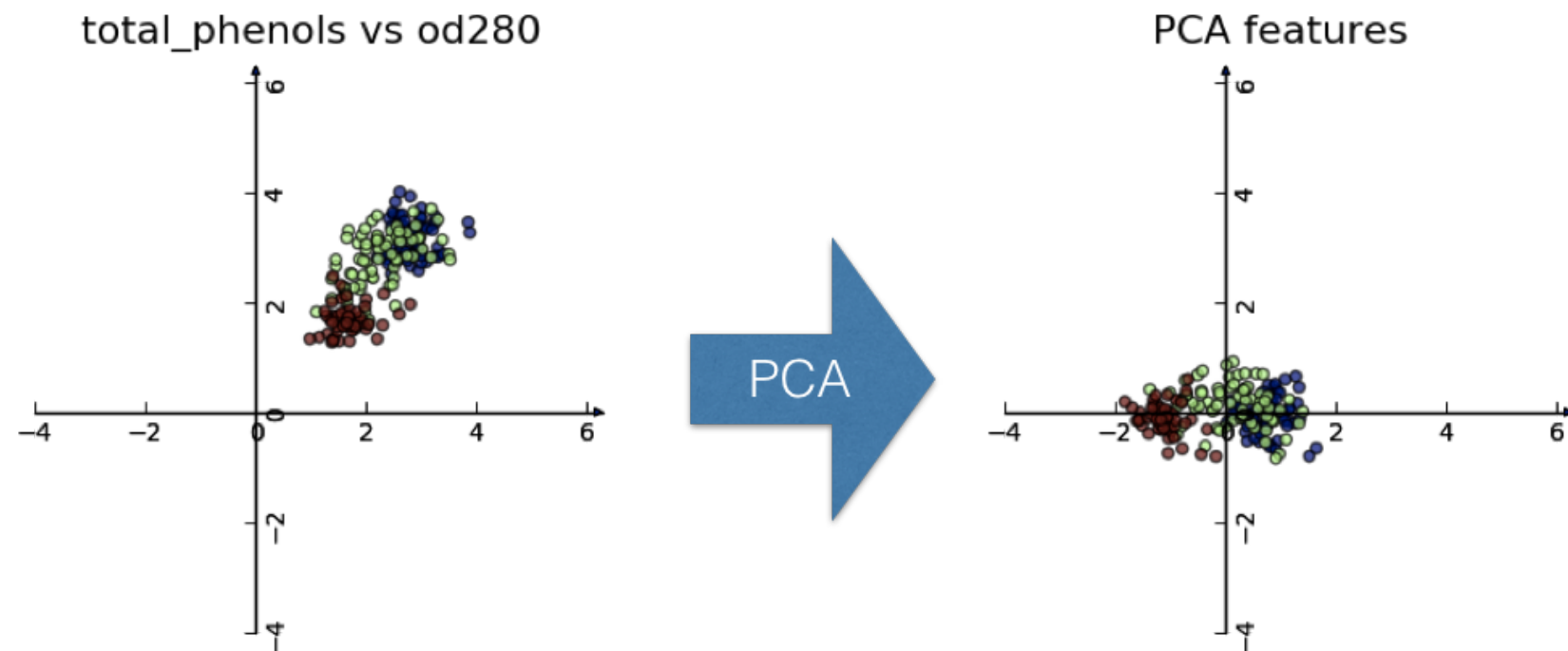
- Rows of transformed correspond to samples
- Columns of transformed are the "PCA features"
- Row gives PCA feature values of corresponding sample

```
print(transformed)
```

```
[[ 1.32771994e+00  4.51396070e-01]
 [ 8.32496068e-01  2.33099664e-01]
 ...
 [-9.33526935e-01 -4.60559297e-01]]
```

# PCA features are not correlated

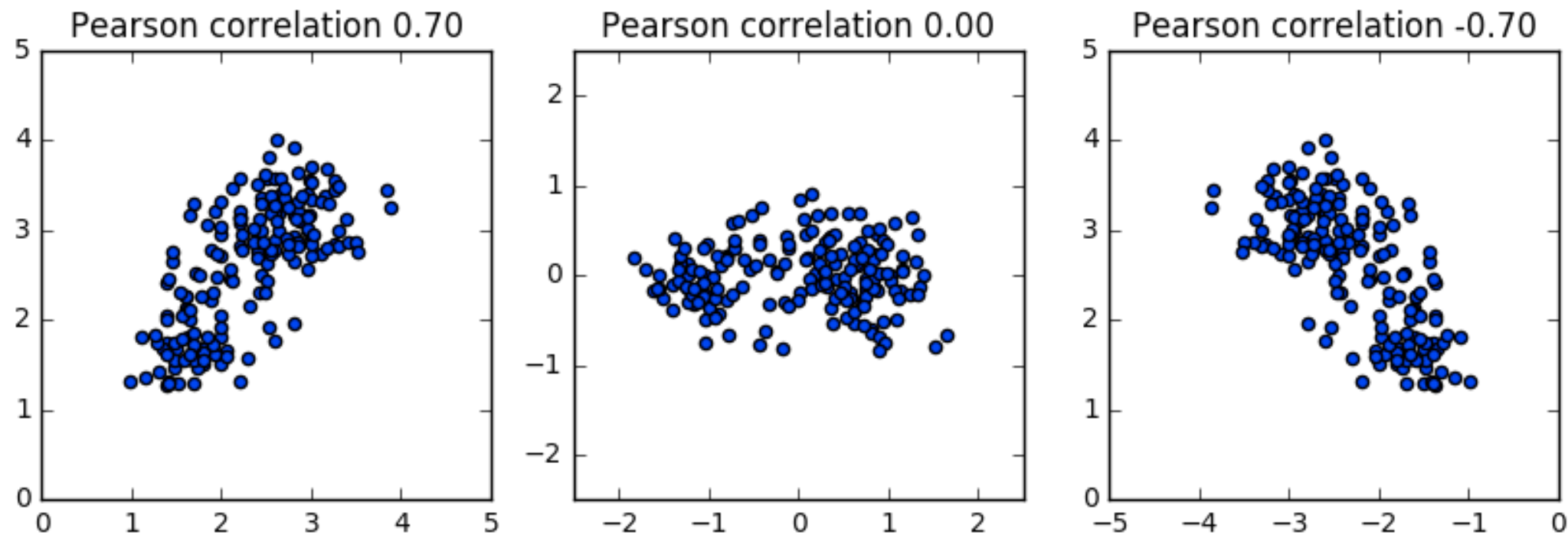
- Features of dataset are often correlated, e.g. `total_phenols` and `od280`
- PCA aligns the data with axes
- Resulting PCA features are not linearly correlated ("decorrelation")





# Pearson correlation

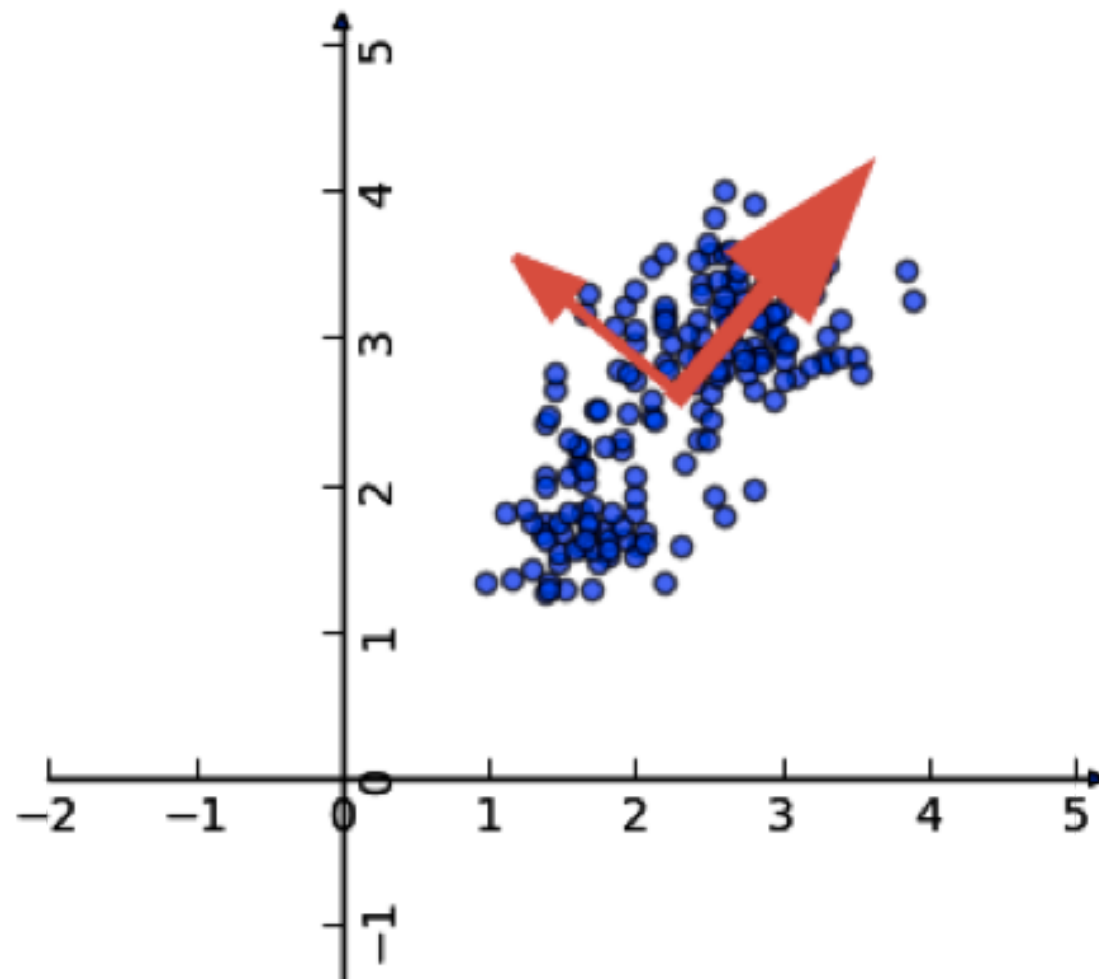
- Measures linear correlation of features
- Value between -1 and 1
- Value of 0 means no linear correlation



# Principal components

- "Principal components" = directions of variance
- PCA aligns principal components with the axes

The Principal Components



# Principal components

- Available as `components_` attribute of PCA object
- Each row defines displacement from mean

```
print(model.components_)
```

```
[[ 0.64116665  0.76740167]  
 [-0.76740167  0.64116665]]
```

# Let's practice!

UNSUPERVISED LEARNING IN PYTHON

# Intrinsic dimension

UNSUPERVISED LEARNING IN PYTHON



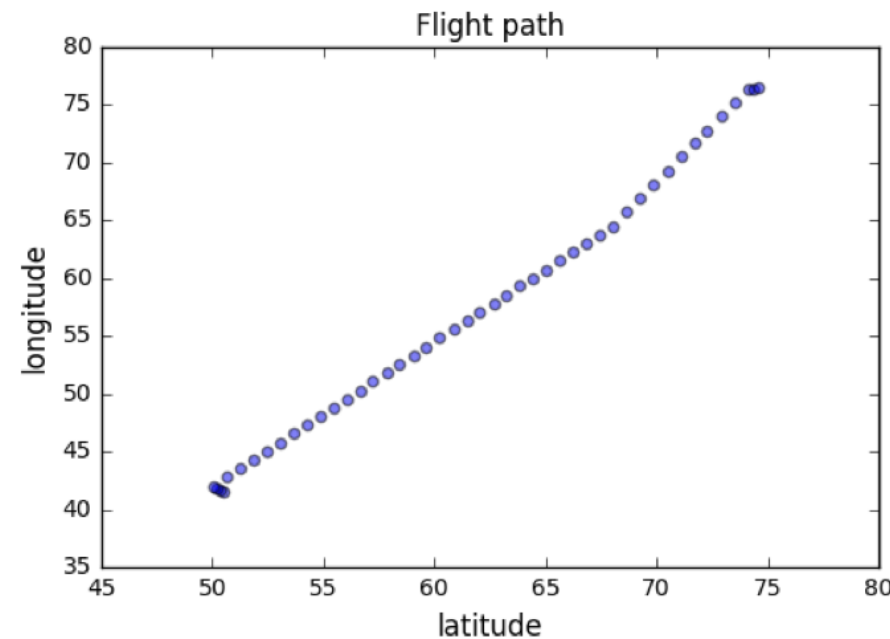
**Benjamin Wilson**

Director of Research at lateral.io

# Intrinsic dimension of a flight path

- 2 features: longitude and latitude at points along a flight path
- Dataset *appears* to be 2-dimensional
- But can approximate using one feature: displacement along flight path
- Is intrinsically 1-dimensional

```
latitude longitude
50.529      41.513
50.360      41.672
50.196      41.835
...
```



# Intrinsic dimension

- Intrinsic dimension = number of features needed to approximate the dataset
- Essential idea behind dimension reduction
- What is the most compact representation of the samples?
- Can be detected with PCA

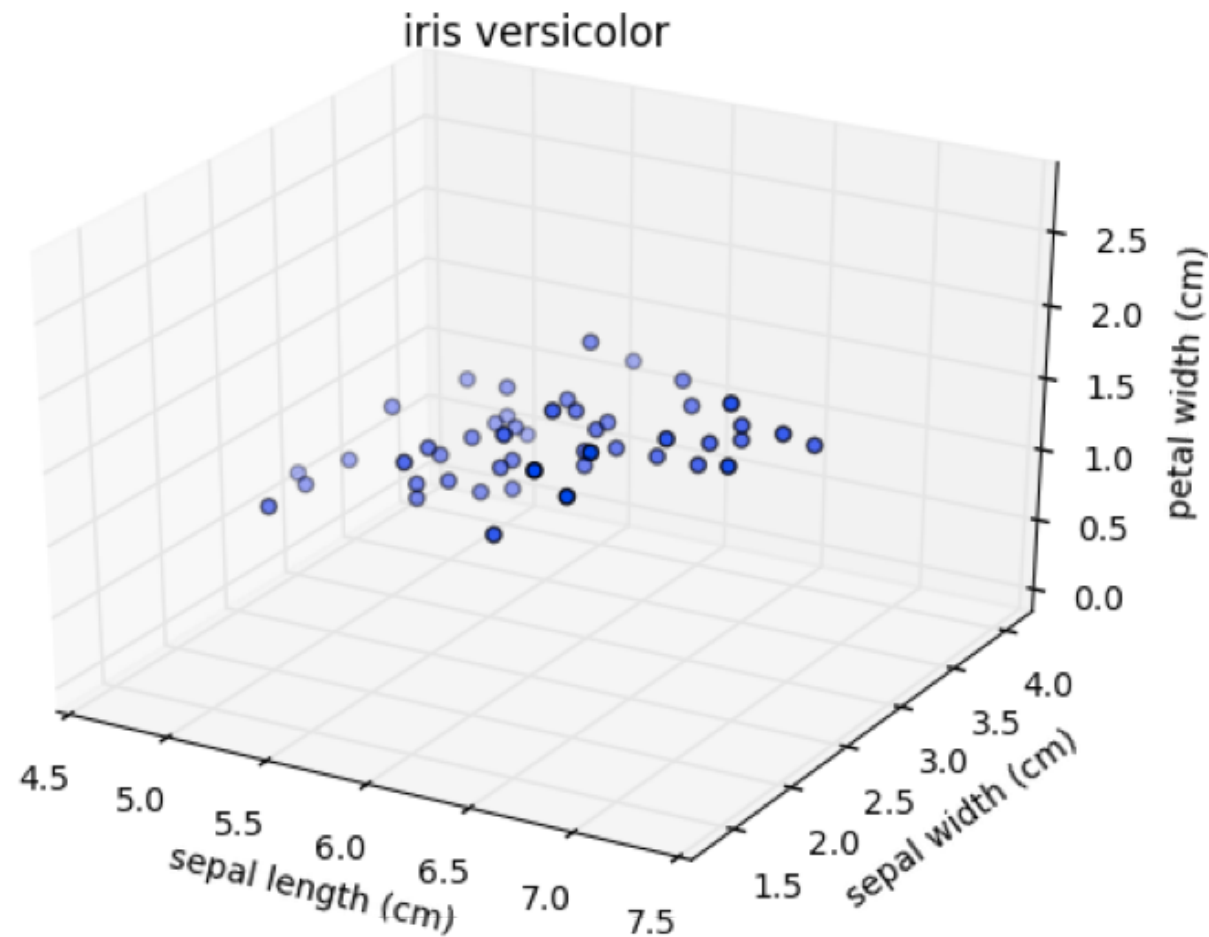
# Versicolor dataset

- "versicolor", one of the iris species
- Only 3 features: sepal length, sepal width, and petal width
- Samples are points in 3D space



# Versicolor dataset has intrinsic dimension 2

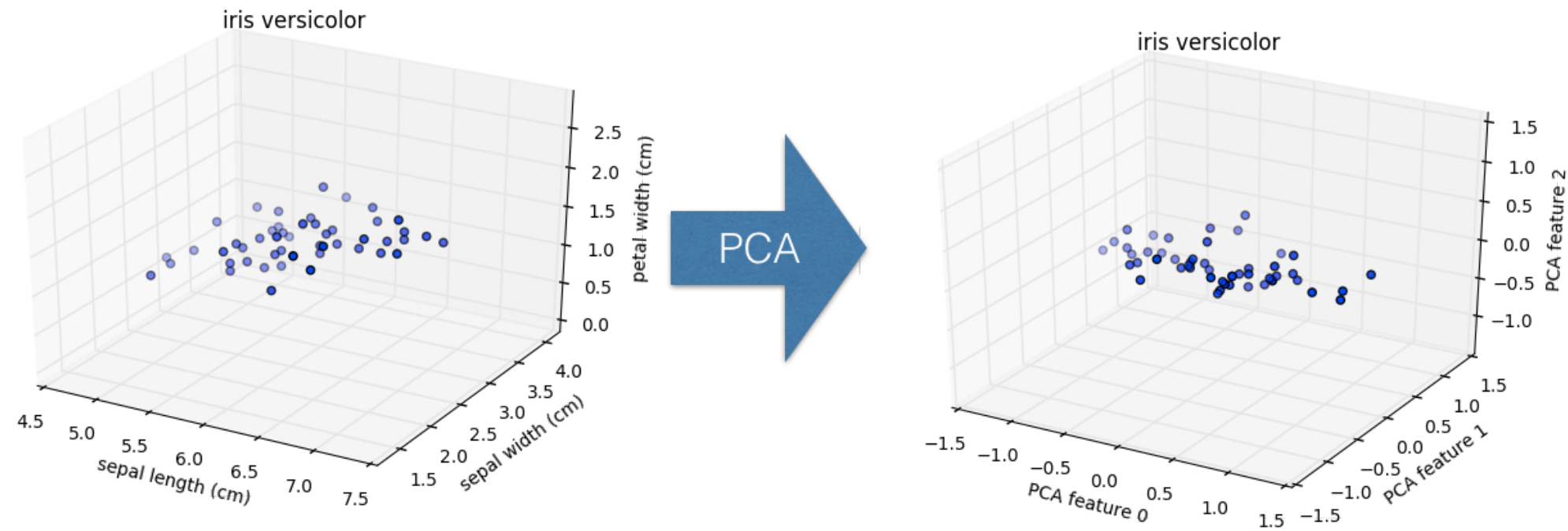
- Samples lie close to a flat 2-dimensional sheet
- So can be approximated using 2 features



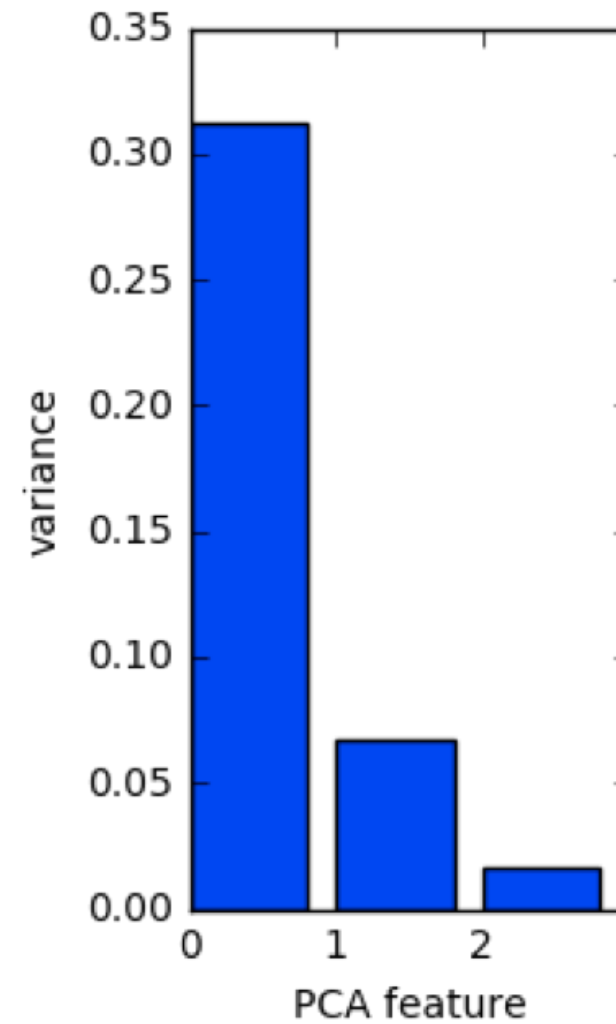
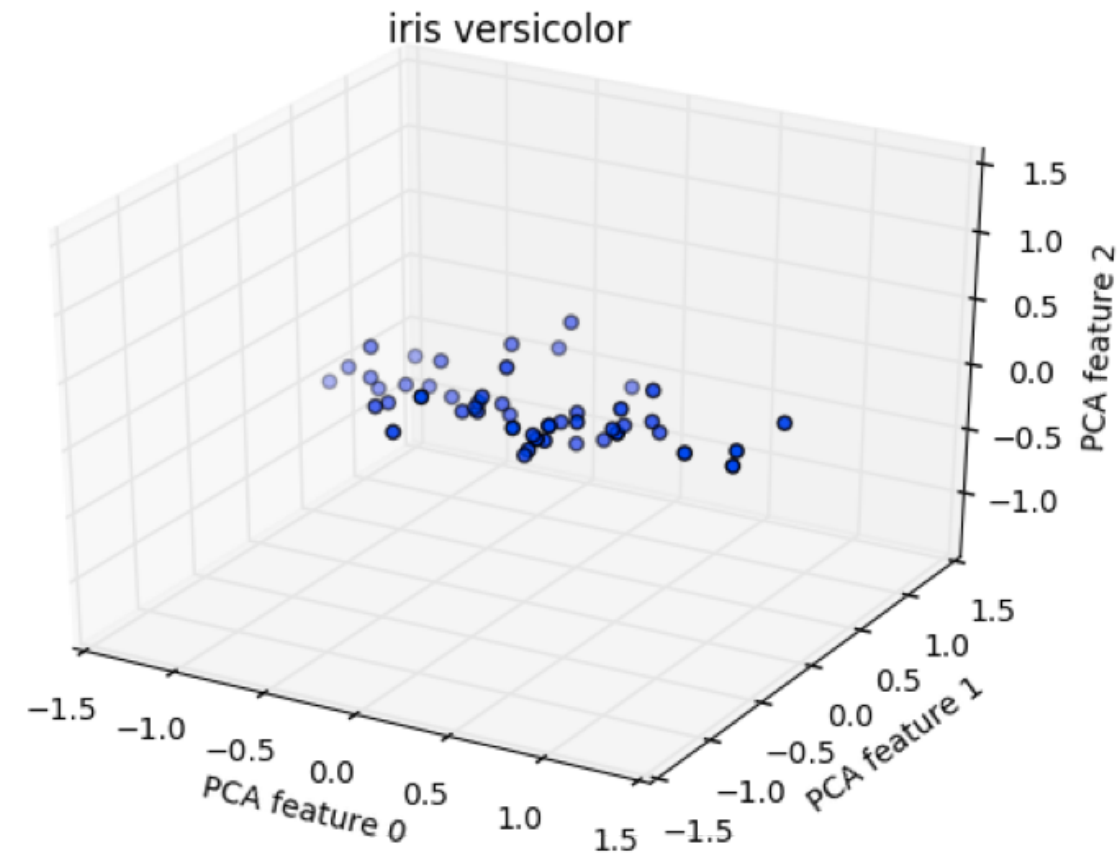
# PCA identifies intrinsic dimension

- Scatter plots work only if samples have 2 or 3 features
- PCA identifies intrinsic dimension when samples have any number of features
- Intrinsic dimension = number of PCA features with significant variance

# PCA of the versicolor samples

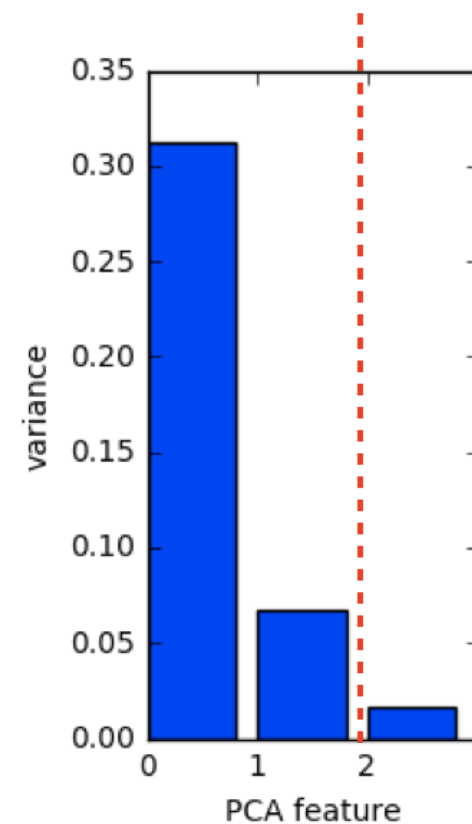


# PCA features are ordered by variance descending



# Variance and intrinsic dimension

- Intrinsic dimension is number of PCA features with significant variance
- In our example: the first two PCA features
- So intrinsic dimension is 2



# Plotting the variances of PCA features

- `samples` = array of versicolor samples

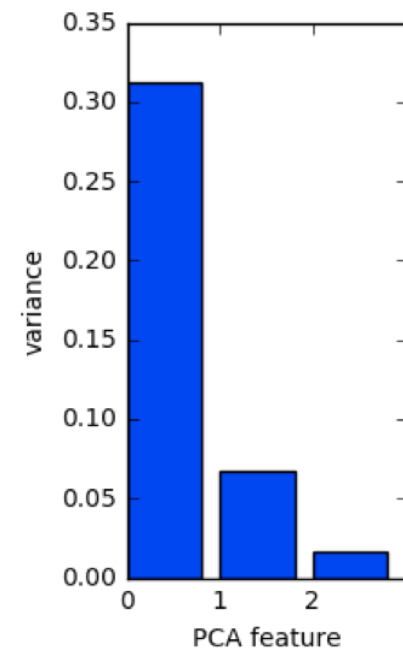
```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(samples)
```

```
PCA()
```

```
features = range(pca.n_components_)
```

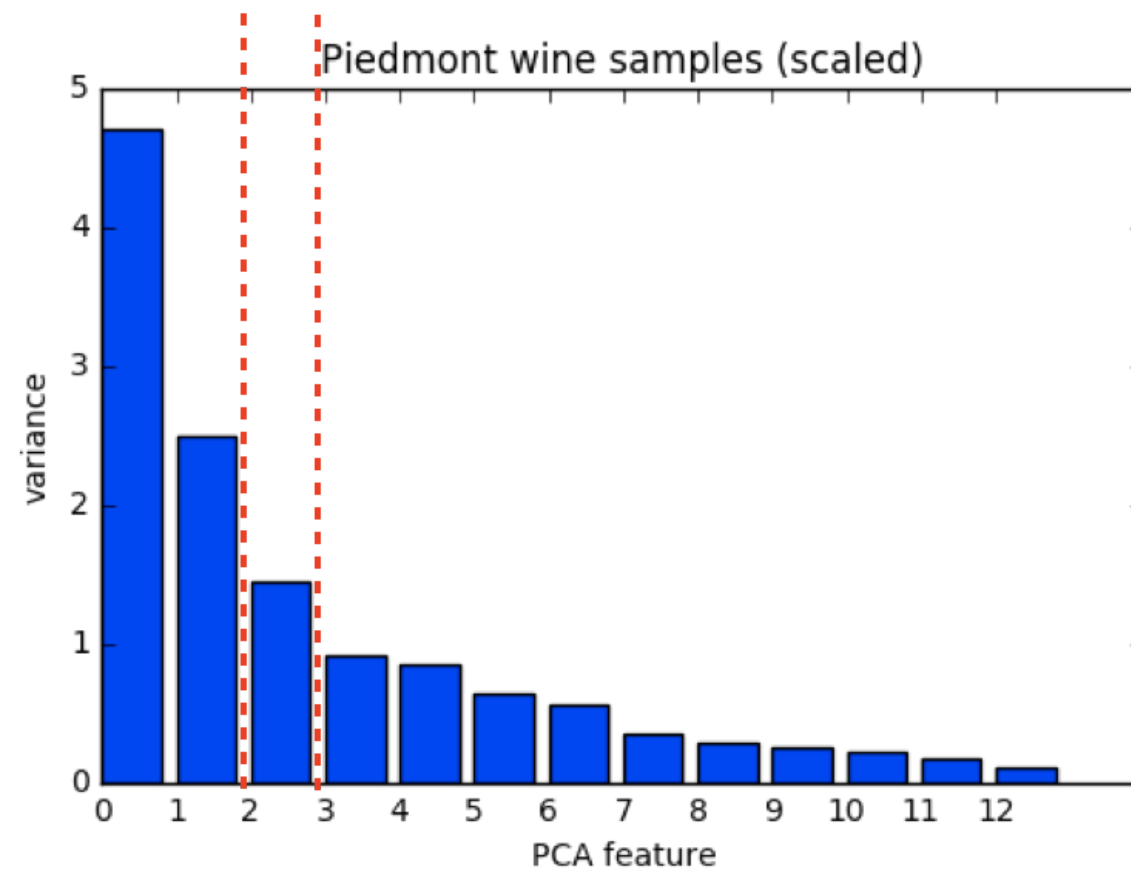
# Plotting the variances of PCA features

```
plt.bar(features, pca.explained_variance_)  
plt.xticks(features)  
plt.ylabel('variance')  
plt.xlabel('PCA feature')  
plt.show()
```



# Intrinsic dimension can be ambiguous

- Intrinsic dimension is an idealization
- ... there is not always one correct answer!
- Piedmont wines: could argue for 2, or for 3, or more





# Let's practice!

UNSUPERVISED LEARNING IN PYTHON

# Dimension reduction with PCA

UNSUPERVISED LEARNING IN PYTHON



**Benjamin Wilson**

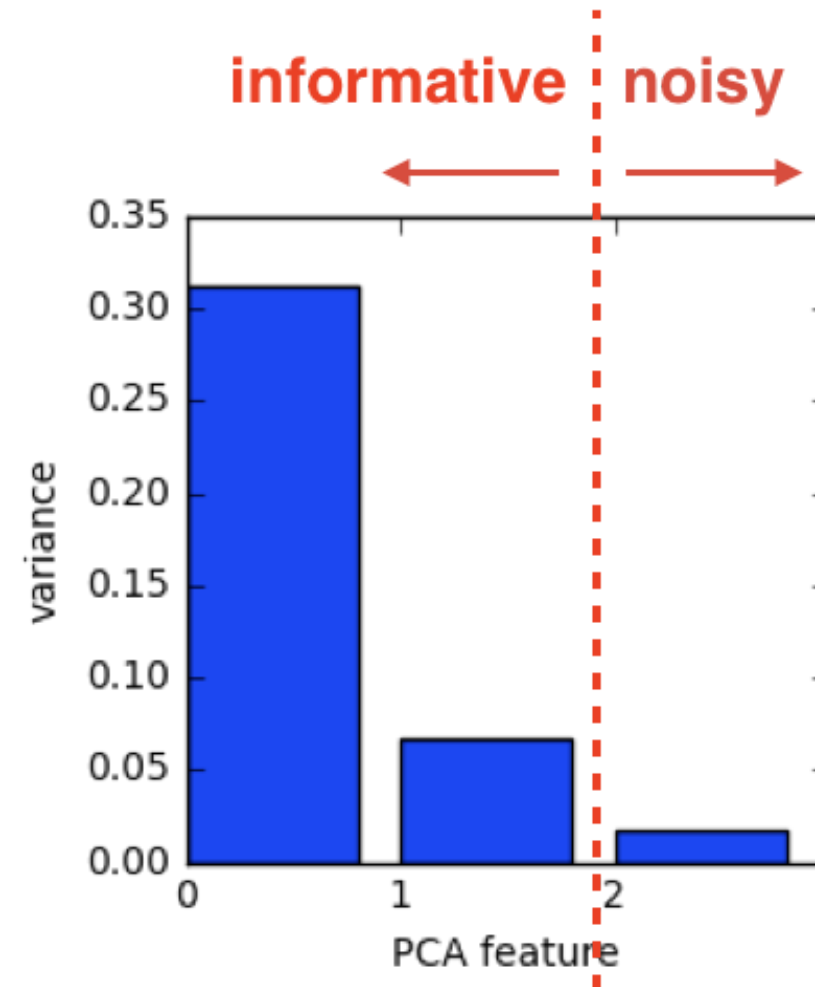
Director of Research at lateral.io

# Dimension reduction

- Represents same data, using less features
- Important part of machine-learning pipelines
- Can be performed using PCA

# Dimension reduction with PCA

- PCA features are in decreasing order of variance
- Assumes the low variance features are "noise"
- ... and high variance features are informative



# Dimension reduction with PCA

- Specify how many features to keep
- E.g. `PCA(n_components=2)`
- Keeps the first 2 PCA features
- Intrinsic dimension is a good choice

# Dimension reduction of iris dataset

- `samples` = array of iris measurements (4 features)
- `species` = list of iris species numbers

```
from sklearn.decomposition import PCA  
pca = PCA(n_components=2)  
pca.fit(samples)
```

```
PCA(n_components=2)
```

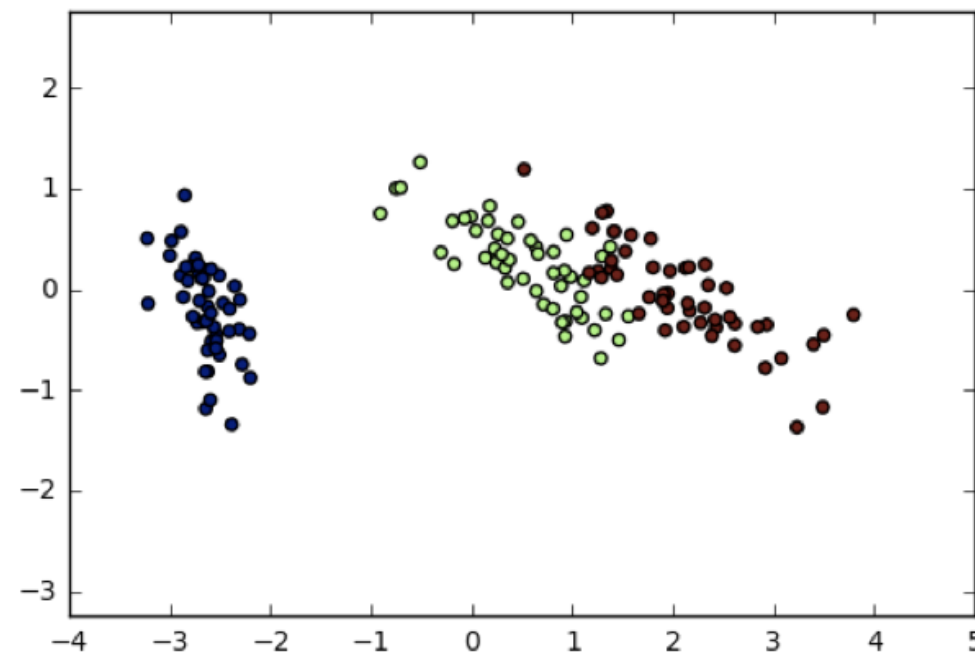
```
transformed = pca.transform(samples)  
print(transformed.shape)
```

```
(150, 2)
```

# Iris dataset in 2 dimensions

- PCA has reduced the dimension to 2
- Retained the 2 PCA features with highest variance
- Important information preserved: species remain distinct

```
import matplotlib.pyplot as plt
xs = transformed[:,0]
ys = transformed[:,1]
plt.scatter(xs, ys, c=species)
plt.show()
```



# Dimension reduction with PCA

- Discards low variance PCA features
- Assumes the high variance features are informative
- Assumption typically holds in practice (e.g. for iris)



# Word frequency arrays

- Rows represent documents, columns represent words
- Entries measure presence of each word in each document
- ... measure using "tf-idf" (more later)

	aardvark	apple	.	.	.	zebra
document0	0,	0.1,	...			0.
document1						
.						
.						
.						
	word frequencies ("tf-idf")					

# Sparse arrays and csr\_matrix

- "Sparse": most entries are zero
- Can use `scipy.sparse.csr_matrix` instead of NumPy array
- `csr_matrix` remembers only the non-zero entries (saves space!)

	aardvark	apple	. . .	zebra
document0	0,	0.1,	...	0.
document1	word frequencies ("tf-idf")			
.				
.				
.				

# TruncatedSVD and csr\_matrix

- scikit-learn `PCA` doesn't support `csr_matrix`
- Use scikit-learn `TruncatedSVD` instead
- Performs same transformation

```
from sklearn.decomposition import TruncatedSVD
model = TruncatedSVD(n_components=3)
model.fit(documents) # documents is csr_matrix
transformed = model.transform(documents)
```

# Let's practice!

UNSUPERVISED LEARNING IN PYTHON