Principal Component Analysis

PHY3287 - Computational Astronomy
Dr Andrea DeMarco
andrea.demarco@um.edu.mt

Data Analysis

- We have seen many classification algorithms (labelled data)
- And also unsupervised estimators to show structure in data (e.g. k-means).
- In this lecture, we explore a very powerful technique - one of the most broadly used unsupervised methods for ‘dimensionality reduction’
- Principal Component Analysis (PCA)
- Also useful for visualisation of data, noise/nuisance filtering, feature extraction/pre-processing etc.

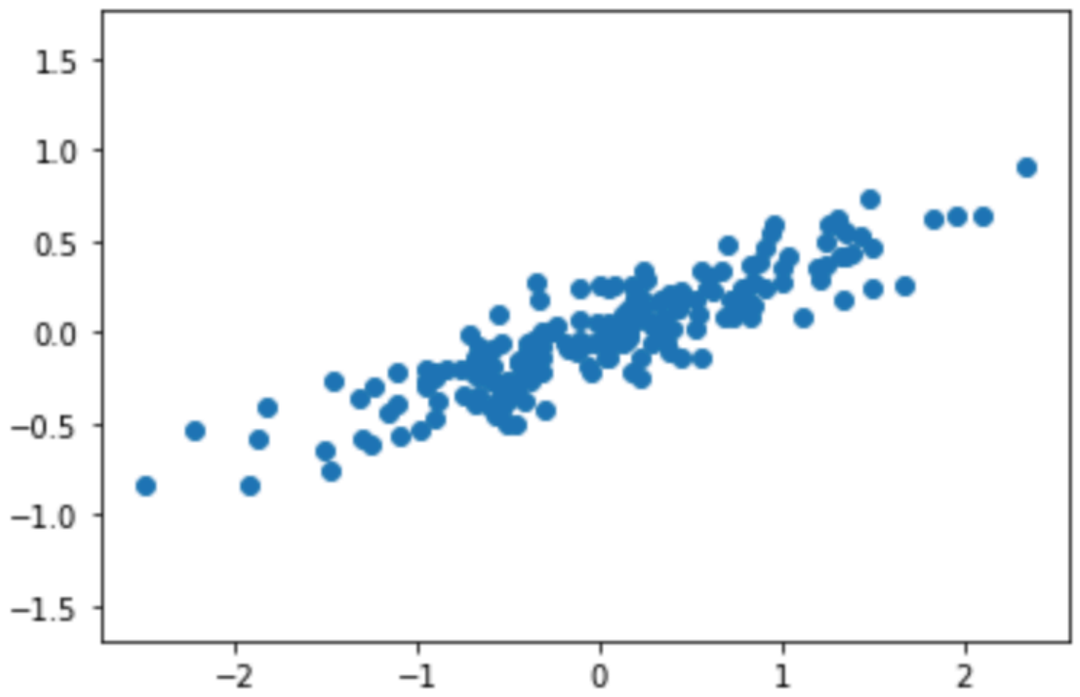
Introducing PCA

- A fast and flexible unsupervised method for dimensionality reduction.

- The PCA concept is best understood visually. Consider a 2D dataset of 200 points

- A clear relationship between x and y

```
rng = np.random.RandomState(1)
X = np.dot(rng.rand(2, 2), rng.randn(2, 200)).T
plt.scatter(X[:, 0], X[:, 1])
plt.axis('equal');
```



Introducing PCA

- Unlike in linear regression, we do not want to predict Y from X, but rather learn the relationship between X and Y
- In PCA, the relationship is quantified by finding a list of principal axes in the data, and using those axes to describe the dataset
- And, we'll use Scikit-Learn's PCA estimator…

Introducing PCA

- The fit learns some qualities from the data:

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

```
PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

- Most importantly the “components” and “explained variance”

```
print(pca.components_)
```

```
[[-0.94446029 -0.32862557]
 [-0.32862557  0.94446029]]
```

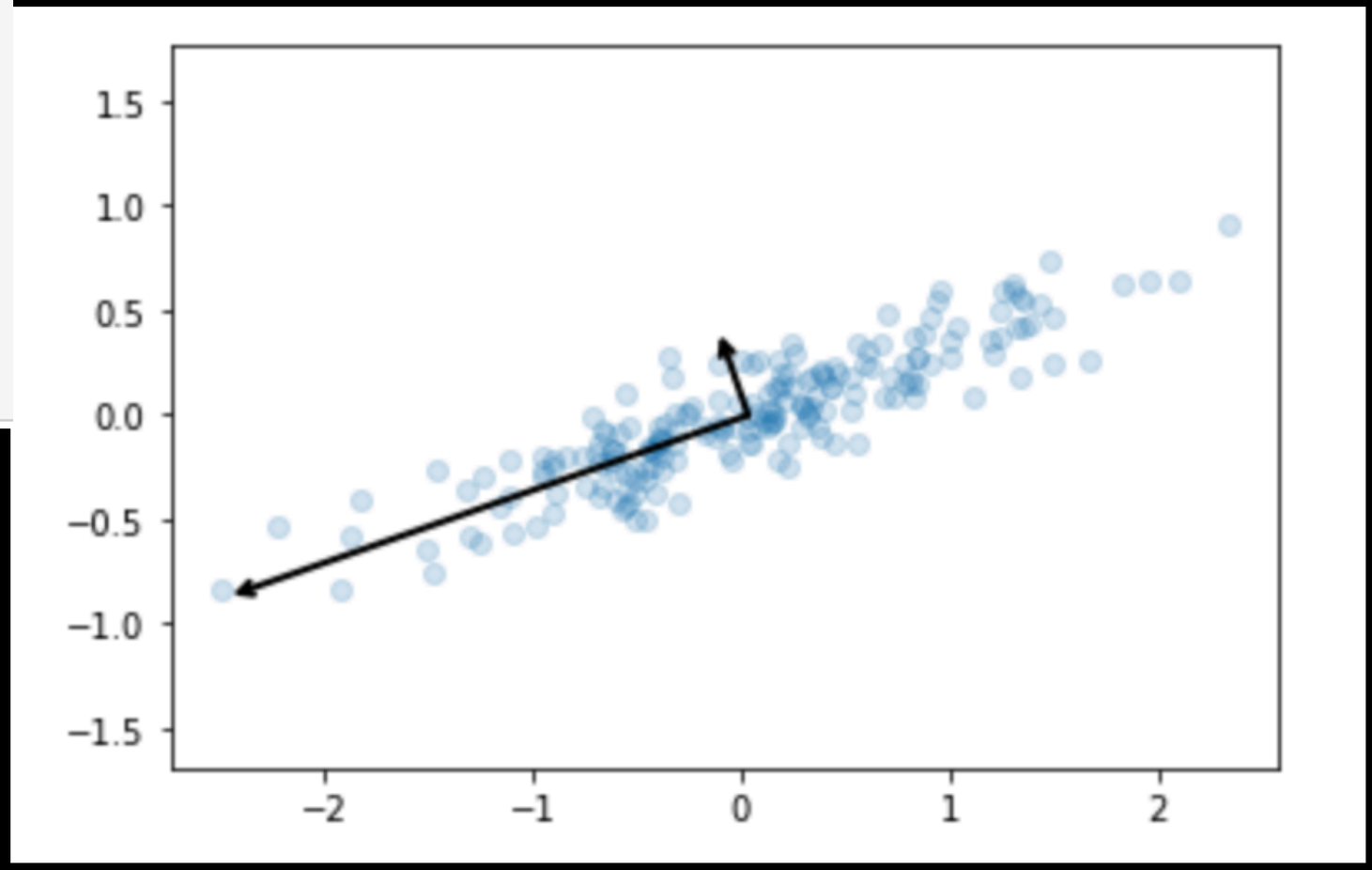
```
print(pca.explained_variance_)
```

```
[0.7625315 0.0184779]
```

Introducing PCA

- Visualising these quantities, we use “components” to define vector direction, and “explained variance” to define the squared-length of the vector:

```
def draw_vector(v0, v1, ax=None):
    ax = ax or plt.gca()
    arrowprops=dict(arrowstyle='->',
                    linewidth=2,
                    shrinkA=0, shrinkB=0)
    ax.annotate('', v1, v0, arrowprops=arrowprops)

# plot data
plt.scatter(X[:, 0], X[:, 1], alpha=0.2)
for length, vector in zip(pca.explained_variance_,
                          pca.components_):
    v = vector * 3 * np.sqrt(length)
    draw_vector(pca.mean_, pca.mean_ + v)
plt.axis('equal');
```



Introducing PCA

- These vectors represent the ‘principal axes’ of the data
- The length of the vector is an indication of how ‘important’ that axis is in describing the distribution of the data - a measure of the variance of the data when projected onto that axis
- The projection of each data point onto the principal axes are the “principal components” of the data
- This transformation from data axes to principal axes is an affine transformation, which basically means it is composed of a translation, rotation, and uniform scaling.

PCA
Dimensionality Reduction

- Using PCA for dimensionality reduction involves zeroing out one or more of the smallest principal components

- Resulting in a lower-dimensional projection of the data

- Preserving maximal data variance

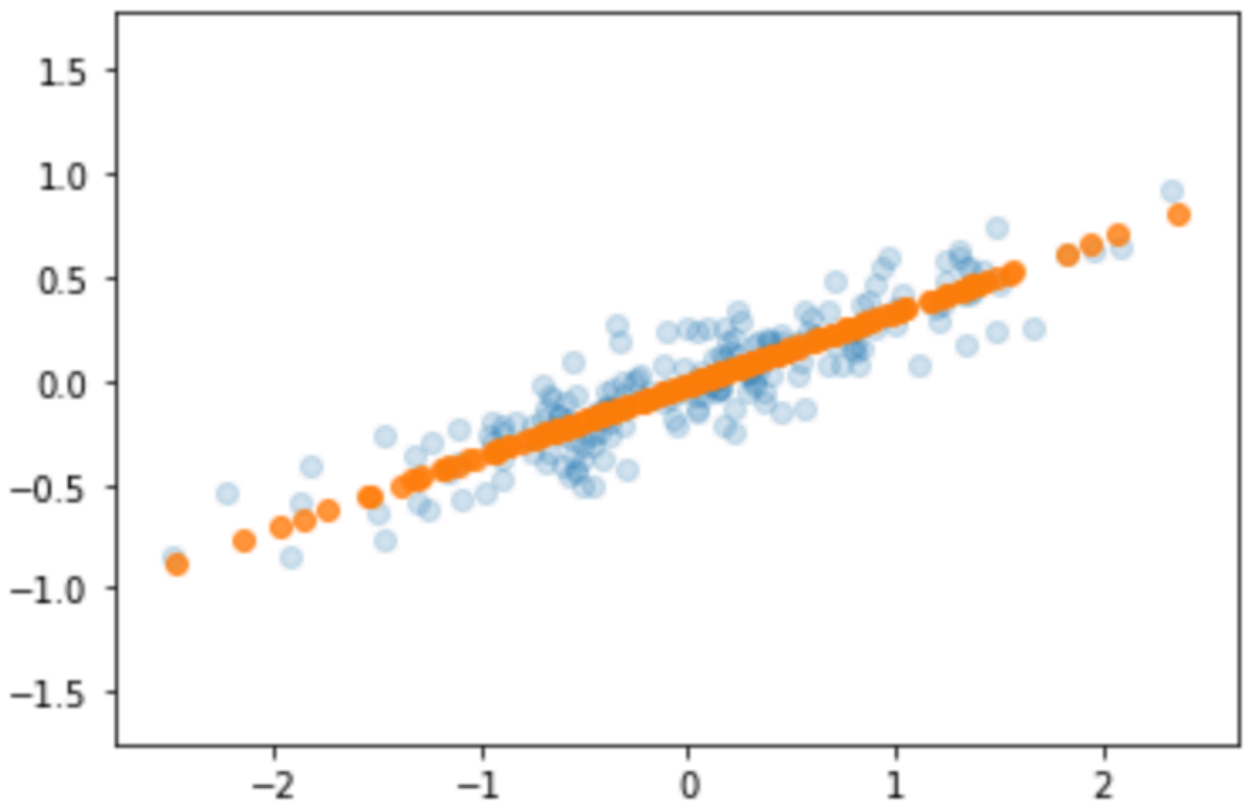
```
pca = PCA(n_components=1)
pca.fit(X)
X_pca = pca.transform(X)
print("original shape:   ", X.shape)
print("transformed shape:", X_pca.shape)
```

```
original shape:    (200, 2)
transformed shape: (200, 1)
```

PCA
Dimensionality Reduction

- Transformed data reduced to just one dimension. If we perform the inverse transform back to a 2D space:

```
X_new = pca.inverse_transform(X_pca)
plt.scatter(X[:, 0], X[:, 1], alpha=0.2)
plt.scatter(X_new[:, 0], X_new[:, 1], alpha=0.8)
plt.axis('equal');
```



PCA for Visualization

- Let's try out a dataset with more dimensions, our RR-Lyrae (4D) dataset

```
# Load up RRLyrae Dataset
data = np.load('./data/rrlyrae.npz')
samples = data['data']
labels = data['labels']
samples.shape
```

```
(93141, 4)
```

- Use PCA to reduce to 2 dimensions:

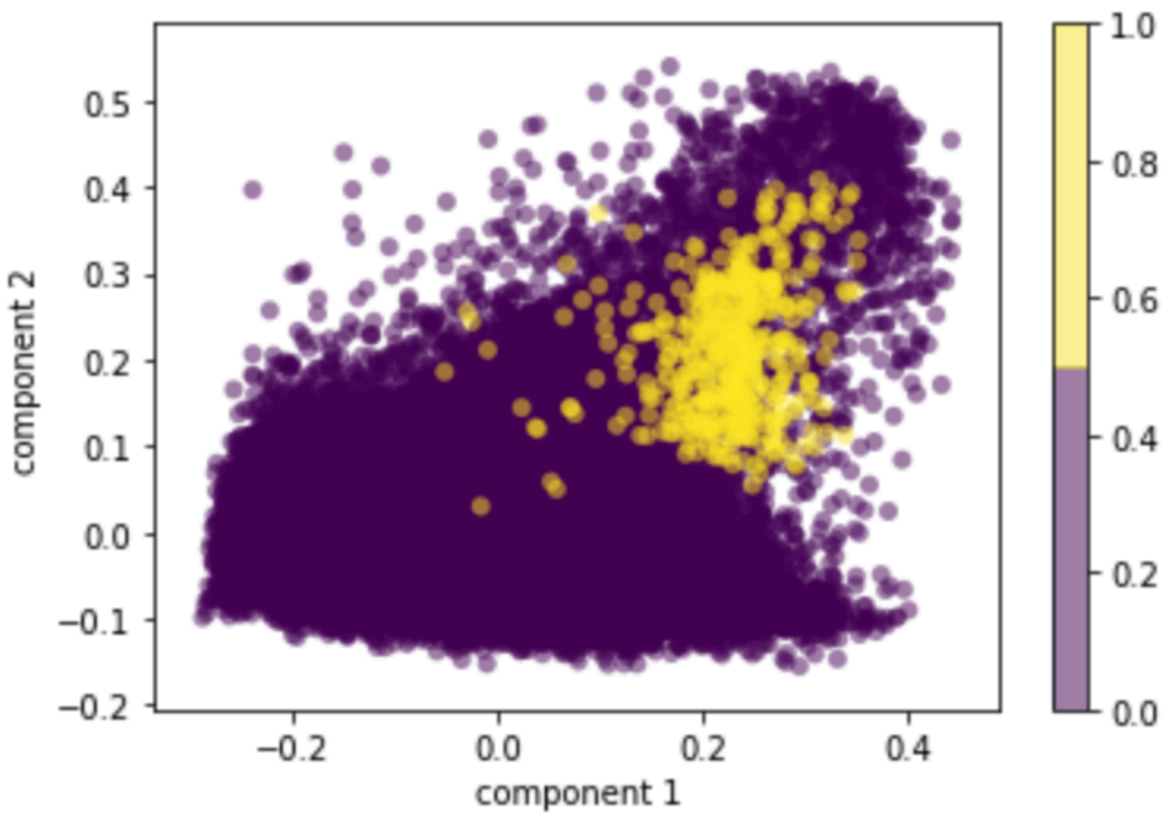
```
pca = PCA(2)  # project from 4 to 2 dimensions
projected = pca.fit_transform(samples)
print(samples.shape)
print(projected.shape)
```

```
(93141, 4)
(93141, 2)
```

PCA for Visualization

- Plot the first two principal components:

```
plt.scatter(projected[:, 0], projected[:, 1],
            c=labels, edgecolor='none', alpha=0.5,
            cmap=plt.cm.get_cmap('viridis', 2))
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.colorbar();
```



PCA for Visualization

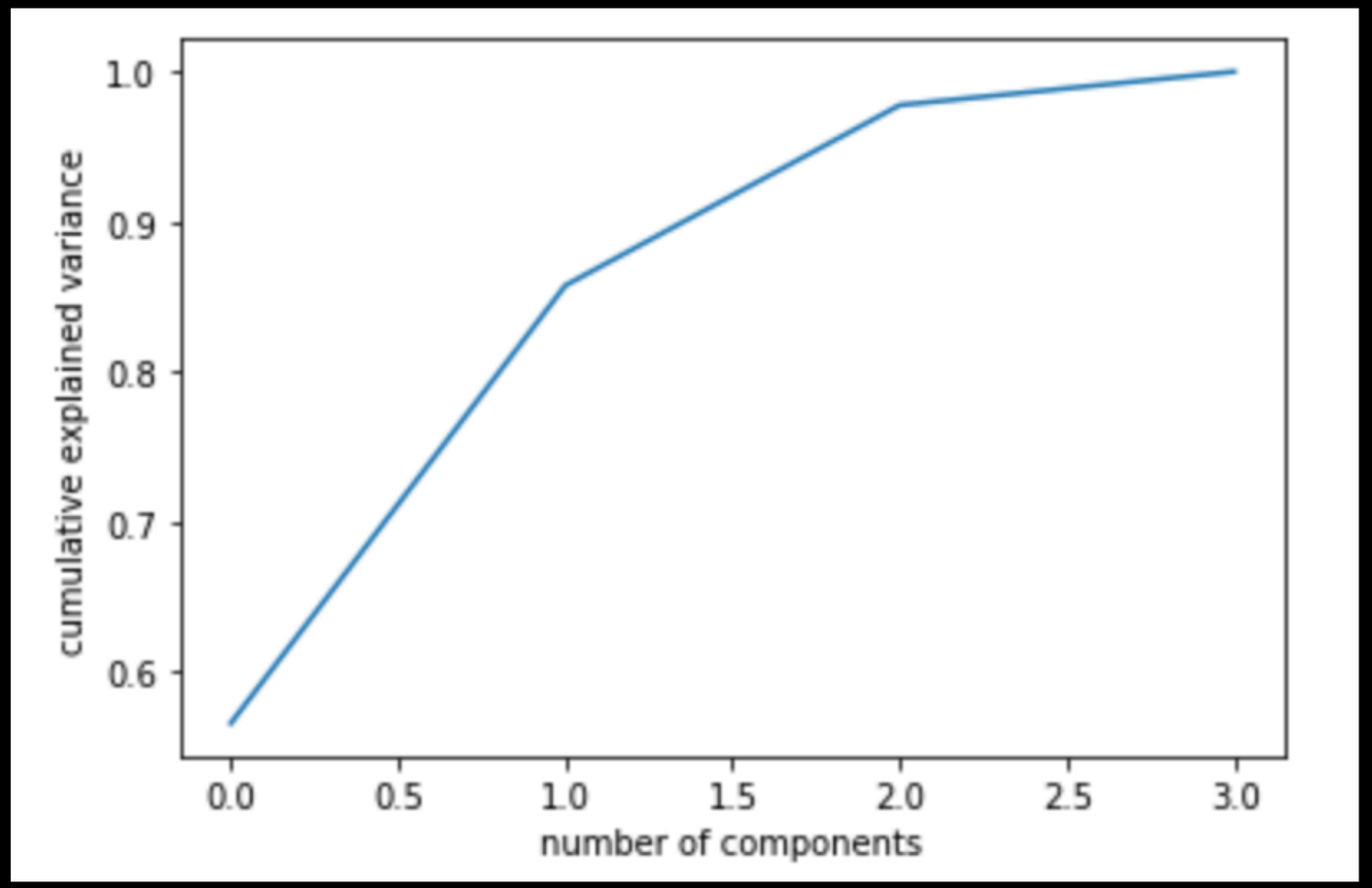
- Recall what these components mean
- The full data is a 4-dimensional point cloud, and these points are the projection of each data point along the directions with the largest variance
- Essentially, we have found the optimal stretch and rotation in 4-dimensional space that allows us to see the layout of the samples in two dimensions, and have done this in an unsupervised manner—that is, without reference to the labels.

Choosing the Number of Components

- A vital part of PCA in practice is to estimate how many components are needed to describe the data

- This can be determined by looking at the cumulative explained variance ratio as a function of the number of components

```
pca = PCA().fit(samples)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```



Case Study: Astronomical Spectra

- The SDSS (Sloan Digital Sky Survey) is a photometric and spectroscopic survey (since 2000)

- A huge astronomical database

- Data contains photometric observations like those we've seen already, but also includes a large number of spectra of various objects

- These spectra are n-dimensional data vectors (n approx 4000)

- Each observation vector (sample) is a flux measurement for a particular wavelength

Case Study: Astronomical Spectra

- Because of the large dimensionality, visualisation is very challenging - this is where PCA may be useful

- We shall not go into the mathematical detail of PCA, but keep in mind that PCA seeks dimensions of the input space which contain the bulk of the variability

Case Study: Astronomical Spectra

- The model has this form: $\overrightarrow{x_i} = \overrightarrow{\mu} + \sum_{j=1}^{n} a_{ij} \overrightarrow{v_j}$

- $\overrightarrow{x_i}$ represents an individual spectrum

- $\overrightarrow{\mu}$ is the mean spectrum for the dataset

- The remaining term encodes the contributions of each of the eigenvectors $\overrightarrow{v_j}$

- The eigenvectors are generally arranged so that those with the smallest $j$contain the most signal-to-noise, and are the most important vectors in reconstructing the spectra.

- For this reason, truncating the sum at some $m < n$ can still result in a faithful representation of the input.

Case Study: Astronomical Spectra

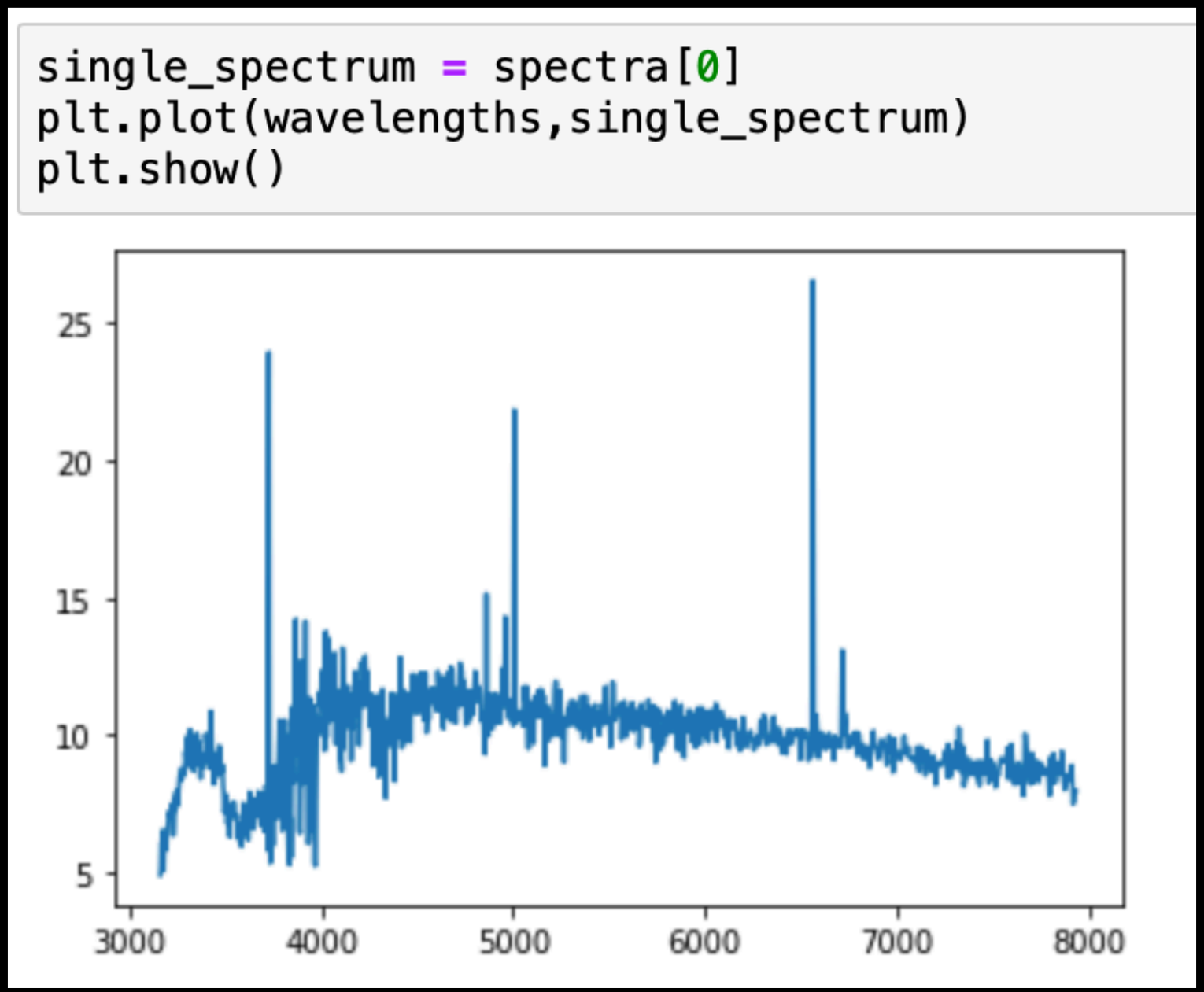
```
def reconstruct_spectra(data):
    spectra = data['spectra']
    coeffs = data['coeffs']
    evecs = data['evecs']
    mask = data['mask']
    mu = data['mu']
    norms = data['norms']
    spec_recons = spectra.copy()
    nev = coeffs.shape[1]
    spec_fill = mu + np.dot(coeffs, evecs[:nev])
    spec_fill *= norms[:, np.newaxis]
    spec_recons[mask] = spec_fill[mask]
    return spec_recons


def compute_wavelengths(data):
    return 10 ** (data['coeff0']
                  + data['coeff1'] * np.arange(data['spectra'].shape[1]))


data = np.load('./data/spec4000.npz')
spectra = reconstruct_spectra(data)
wavelengths = compute_wavelengths(data)

print(spectra.shape)
print(wavelengths.shape)
```

```
(4000, 1000)
(1000,)
```

Case Study: Astronomical Spectra

- Plotting a single spectrum:

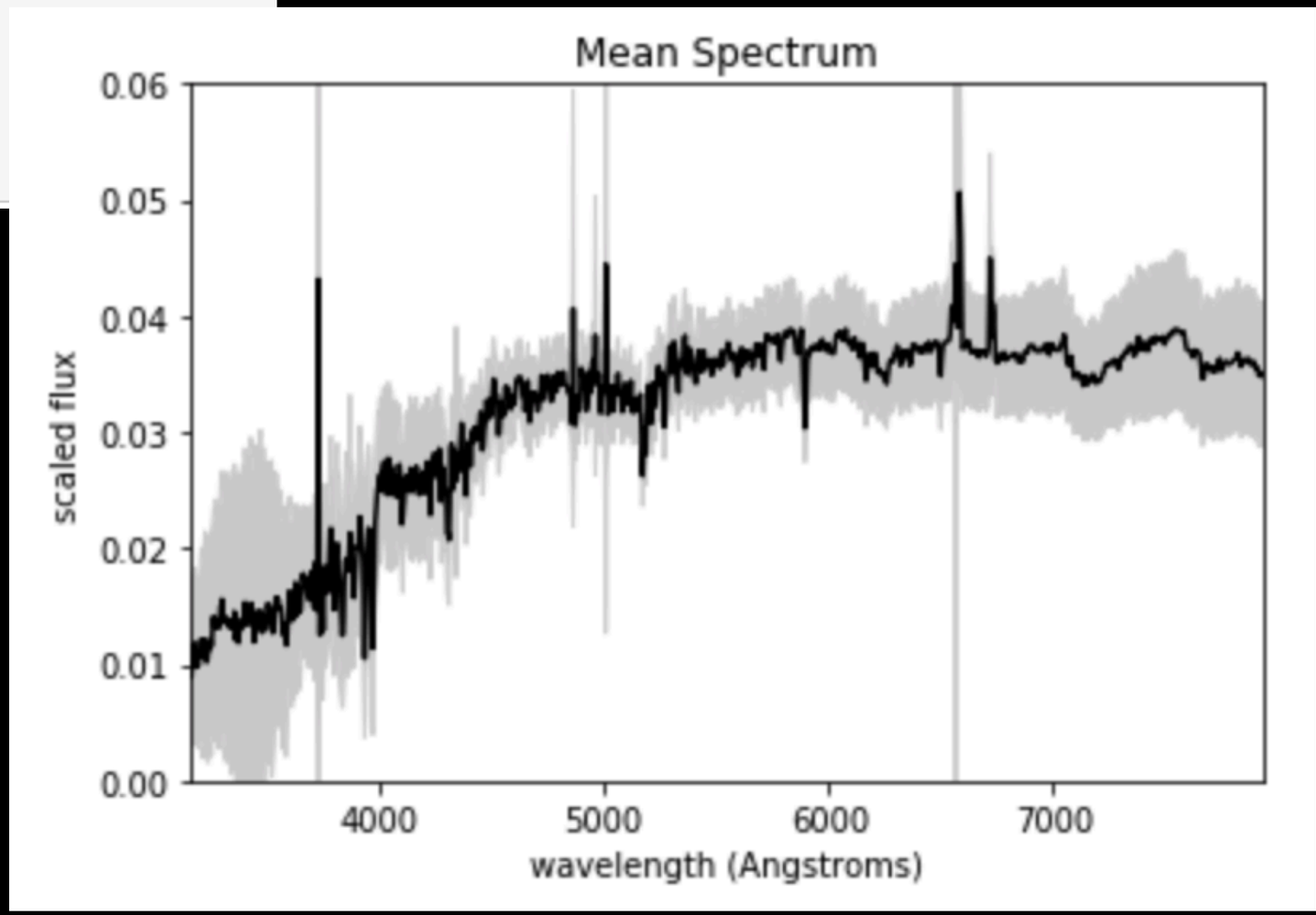
```
single_spectrum = spectra[0]
plt.plot(wavelengths,single_spectrum)
plt.show()
```



Case Study: Astronomical Spectra

- Plotting the mean spectrum:

```
from sklearn import preprocessing
spectra = preprocessing.normalize(spectra)
mu = spectra.mean(0)
std = spectra.std(0)
plt.plot(wavelengths, mu, color='black')
plt.fill_between(wavelengths, mu - std, mu + std, color='#CCCCCC')
plt.xlim(wavelengths[0], wavelengths[-1])
plt.ylim(0, 0.06)
plt.xlabel('wavelength (Angstroms)')
plt.ylabel('scaled flux')
plt.title('Mean Spectrum')
```



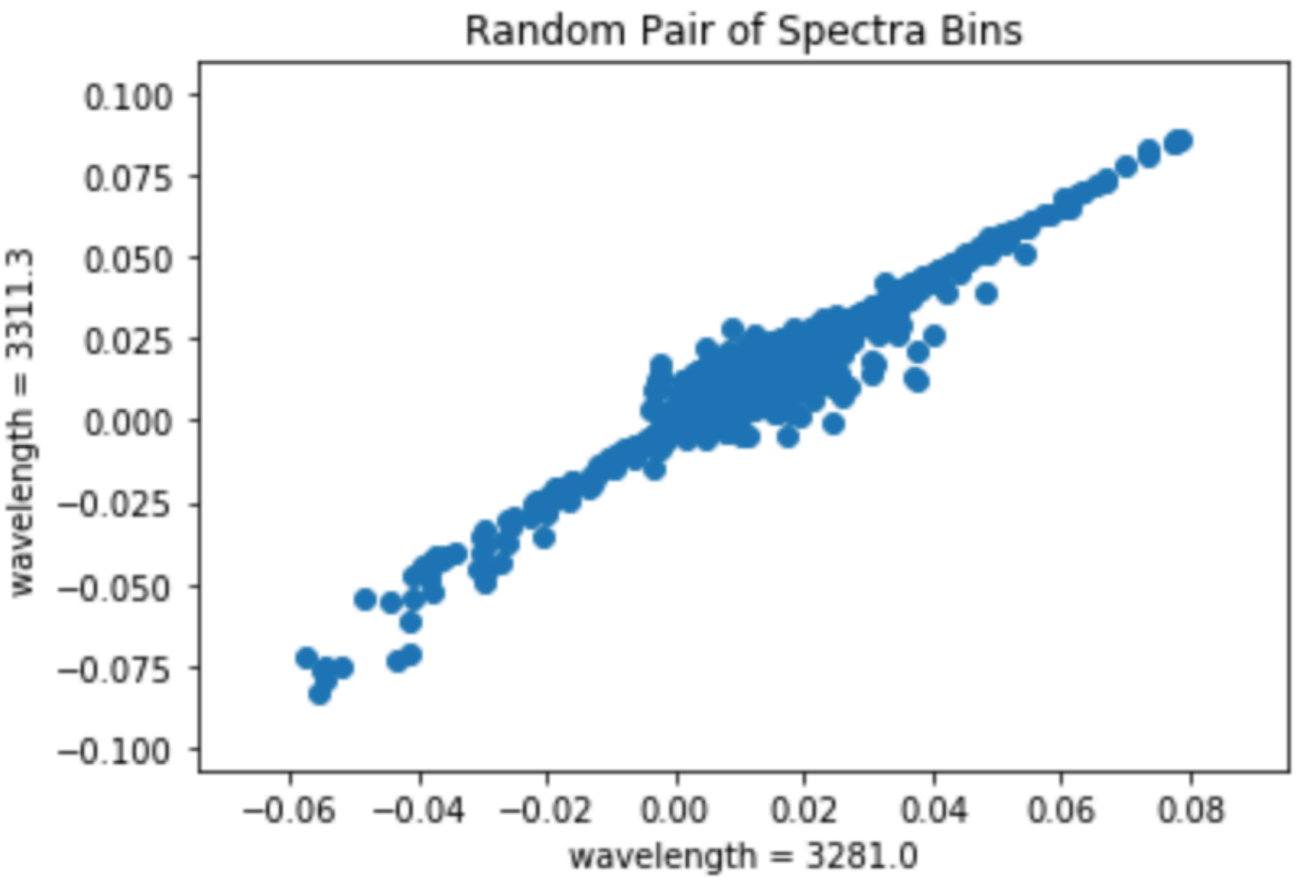
Case Study: Astronomical Spectra

- The interesting part of the data is the gray shaded regions: how do spectra vary from the mean, and how can this variation tell us about their physical properties?
- One option to visualize this would be to scatter-plot random pairs of observations from each spectrum.

Case Study: Astronomical Spectra

```
np.random.seed(25255)  # this seed is chosen to emphasize correlation
i1, i2 = np.random.randint(1000, size=2)
plt.scatter(spectra[:, i1], spectra[:, i2])
plt.xlabel('wavelength = %.1f' % wavelengths[i1])
plt.ylabel('wavelength = %.1f' % wavelengths[i2])
plt.title('Random Pair of Spectra Bins')
```

```
Text(0.5, 1.0, 'Random Pair of Spectra Bins')
```



Case Study: Astronomical Spectra

- There is clear correlation between these 2 measurements. If you know one, you can predict the other.

- So, some spectral bins do not add much information

- We could proceed to analyse all possible pairs of spectral bins, plotting the attributes and deciding on which bins can be removed.

- But that would be very tedious…

- So, PCA

Case Study: Astronomical Spectra

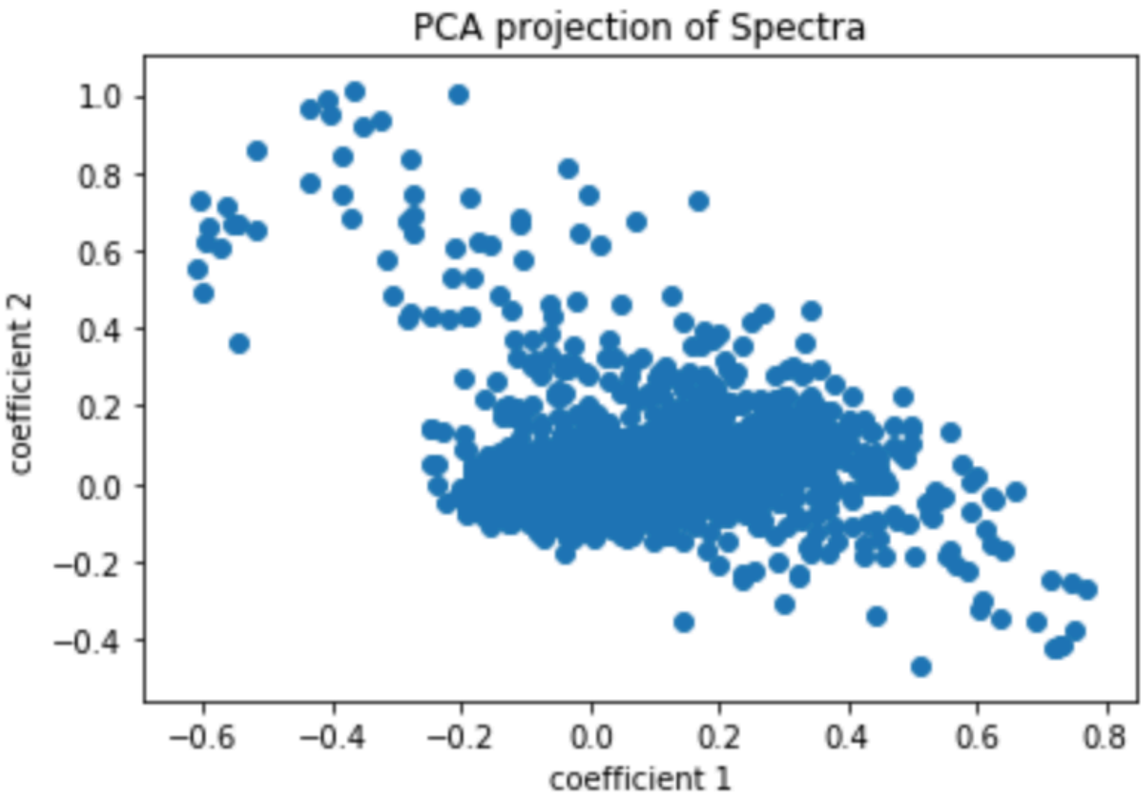
- spectra_projected is now a reduced-dimension representation (down to 4 from 4000!)

```
pca = PCA(4)  # project to just 4 dimensions
spectra_projected = pca.fit_transform(spectra
print(spectra_projected.shape)
```

```
(4000, 4)
```

```
plt.scatter(spectra_projected[:, 0], spectra_projected[:, 1])
plt.xlabel('coefficient 1')
plt.ylabel('coefficient 2')
plt.title('PCA projection of Spectra')
```

```
Text(0.5, 1.0, 'PCA projection of Spectra')
```



Case Study: Astronomical Spectra

- We now have a 2D visualisation, but what is this really showing?

- Going back to our PCA model, we are simply saying that with a limitation in variance of the entire dataset:

$$\overrightarrow{s_i} \approx \overrightarrow{\mu} + a_{i1}\overrightarrow{v_1} + a_{i2}\overrightarrow{v_2}$$

- Each component is associated with an eigenvector, and this plot is showing a_{i1} and a_{i2}

PCA Recap

- Discussed PCA
- Use of dimensionality reduction
- Visualization of high-dimensional data
- PCA is versatile and very interpretable - the model is quite simple
- Making choices about PCA components is also straightforward