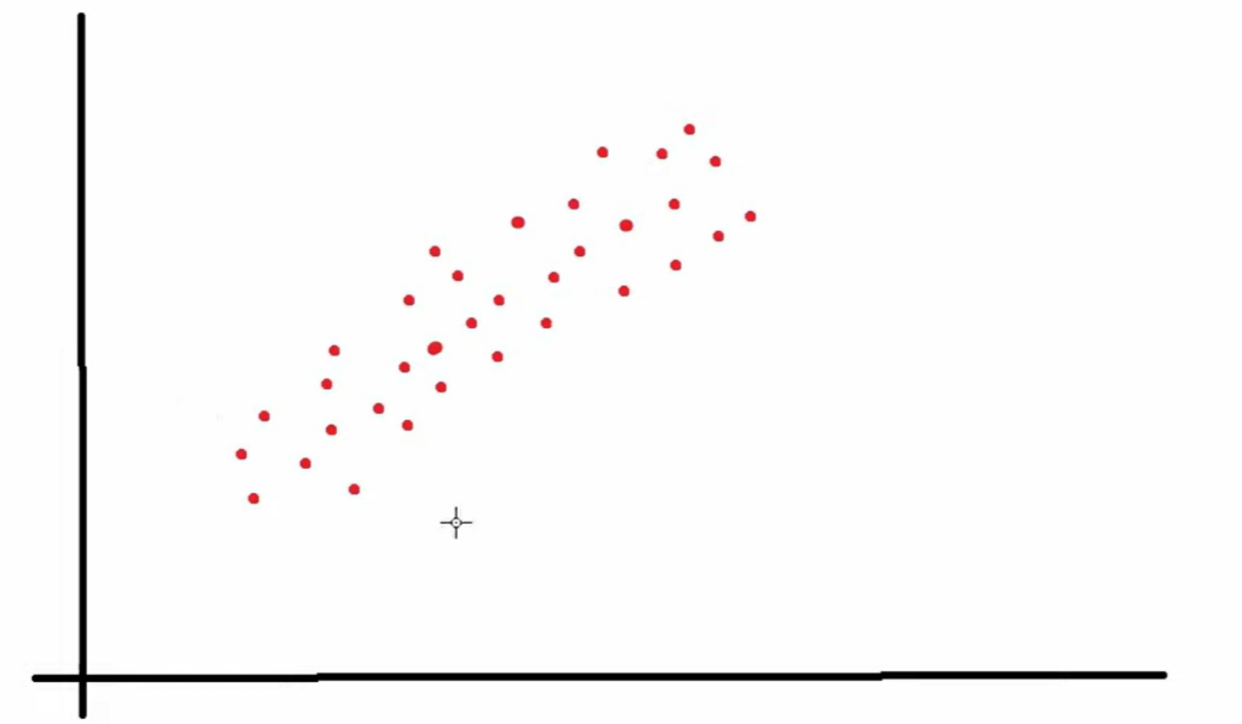
Principle Component Analysis

Ahman Smith

Main principle: Dimensionality reduction

Unsupervised learning algorithm



If we wanted to reduce the dimensionality, we'd project all of these points onto a line.

However, **we want a good line. Not a shitty one.** We find the best line by testing which line can preserve most of the data that we care about.

Takes n-dimensional dataset and finds the best axis that preserves the most variance

An application of PCA: Image Compression

I will apply PCA to this image of a robot in order to compress it:

I begin by importing the necessary libraries and loading the image with the RGB color set:

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
```

✓ 6.9s Python

```python
img = cv2.cvtColor(cv2.imread('C:\\Users\\defra\\Downloads\\download.jpg'), cv2.COLOR_BGR2RGB)
```

✓ 0.0s Python

I then print out the shape of the image just to take a peek:

```python
img.shape
```

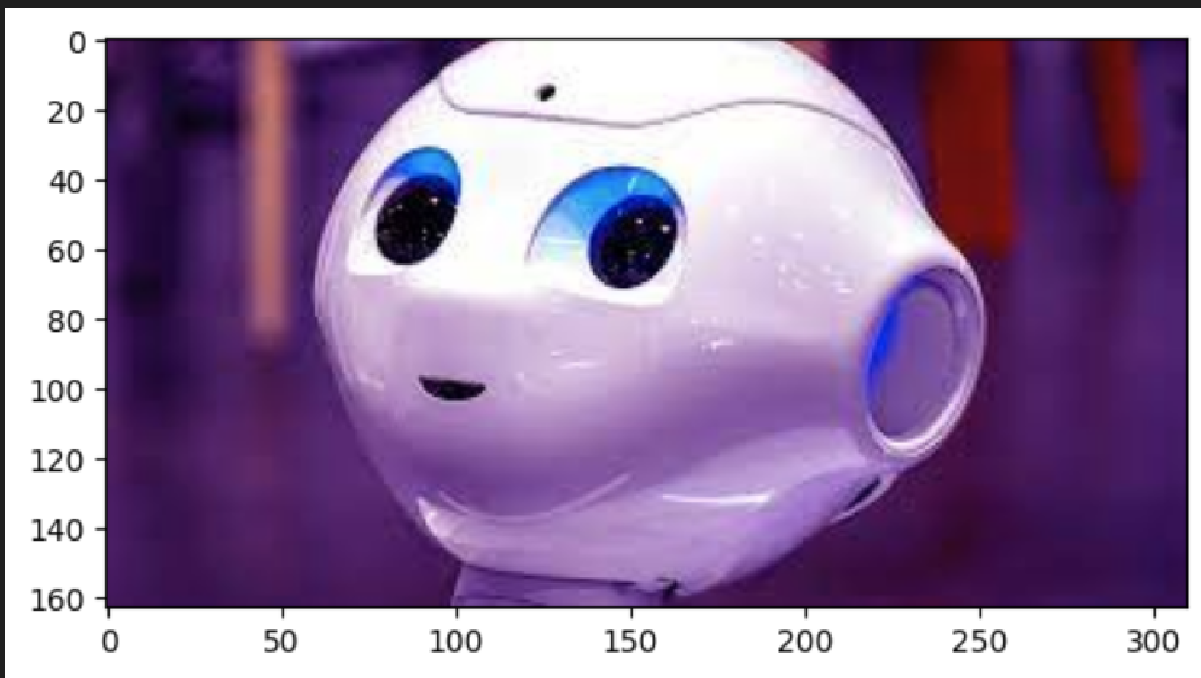
[6] ✓ 0.0s Python

```
(163, 310, 3)
```

163 rows, 310 colums

We can also show the image on a graph by using plt.imshow():

```
<matplotlib.image.AxesImage at 0x21772bc92d0>
```



Reduction

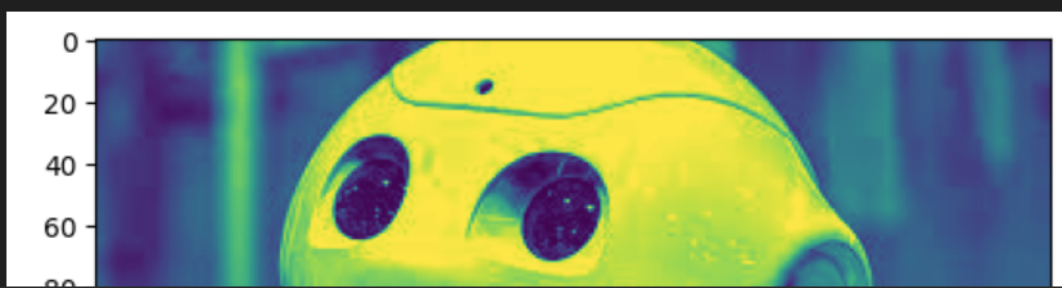
Our image has three channels. We want to split it up.

```
r, g, b = cv2.split(img)
```
[8] ✓ 0.0s

```
r, g, b = r / 255, g / 255, b / 255
```
[9] ✓ 0.1s

```
plt.imshow(r)
```
[10] ✓ 0.5s

```
<matplotlib.image.AxesImage at 0x217726bf2d0>
```



Split it up into R, G, B channels. Divided it all by 255 since that is the maximum value it can have. I printed out the red channel. There isn't much red.

Components

We're going to reduce the components to 25, declared by pca_components = 25

```
pca_components = 25
pca_r = PCA(n_components=pca_components)
reduced_r = pca_r.fit_transform(r)
```
[11] ✓ 0.1s

```
print(reduced_r.shape)
```
[12] ✓ 0.0s

```
(163, 25)
```

As we can see, the columns are reduced.

I did this to every channel.

```
pca_components = 50
pca_r = PCA(n_components=pca_components)
reduced_r = pca_r.fit_transform(r)

pca_g = PCA(n_components=pca_components)
reduced_g = pca_g.fit_transform(g)

pca_b = PCA(n_components=pca_components)
reduced_b = pca_b.fit_transform(b)
```
✓ 0.1s Python

Then, we reconstruct each channel that's reduced:

```
reconstructed_r = pca_r.inverse_transform(reduced_r)
reconstructed_g = pca_g.inverse_transform(reduced_g)
reconstructed_b = pca_b.inverse_transform(reduced_b)
```
✓ 0.0s Python

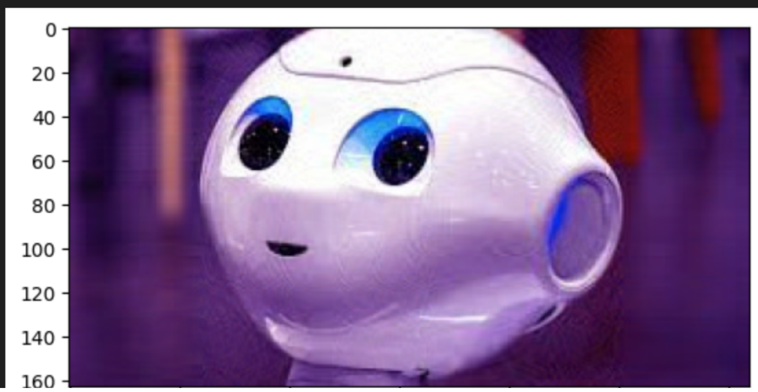
Finally, we can see the output as:

```python
img_reconstructed = (cv2.merge((reconstructed_r, reconstructed_g, reconstructed_b)))
plt.imshow(img_reconstructed)
```

[94] ✓ 0.3s

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

<matplotlib.image.AxesImage at 0x2177a9ae390>
```



Compared to before, there is not much of a difference. This is with 50 principle components.