Lecture 19 - More Applications of PCA, Supervised Dimensionality Reduction & Manifold Learning

Image Compression

Example: Eigenfaces

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('bmh')
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
```

In [3]:
```
from sklearn.datasets import fetch_olivetti_faces

faces = fetch_olivetti_faces(return_X_y=False)
print(faces.DESCR)
```

```
.. _olivetti_faces_dataset:

The Olivetti faces dataset
--------------------------

`This dataset contains a set of face images`_ taken between April 1992 and
April 1994 at AT&T Laboratories Cambridge. The
:func:`sklearn.datasets.fetch_olivetti_faces` function is the data
fetching / caching function that downloads the data
archive from AT&T.

.. _This dataset contains a set of face images: https://cam-orl.co.uk/facedatabase.h
tml

As described on the original website:

    There are ten different images of each of 40 distinct subjects. For some
    subjects, the images were taken at different times, varying the lighting,
    facial expressions (open / closed eyes, smiling / not smiling) and facial
    details (glasses / no glasses). All the images were taken against a dark
    homogeneous background with the subjects in an upright, frontal position
    (with tolerance for some side movement).

**Data Set Characteristics:**

=================   =====================
Classes                                40
Samples total                         400
Dimensionality                       4096
Features            real, between 0 and 1
=================   =====================

The image is quantized to 256 grey levels and stored as unsigned 8-bit
integers; the loader will convert these to floating point values on the
interval [0, 1], which are easier to work with for many algorithms.

The "target" for this database is an integer from 0 to 39 indicating the
identity of the person pictured; however, with only 10 examples per class, this
relatively small dataset is more interesting from an unsupervised or
semi-supervised perspective.

The original dataset consisted of 92 x 112, while the version available here
consists of 64x64 images.

When using these images, please give credit to AT&T Laboratories Cambridge.
```

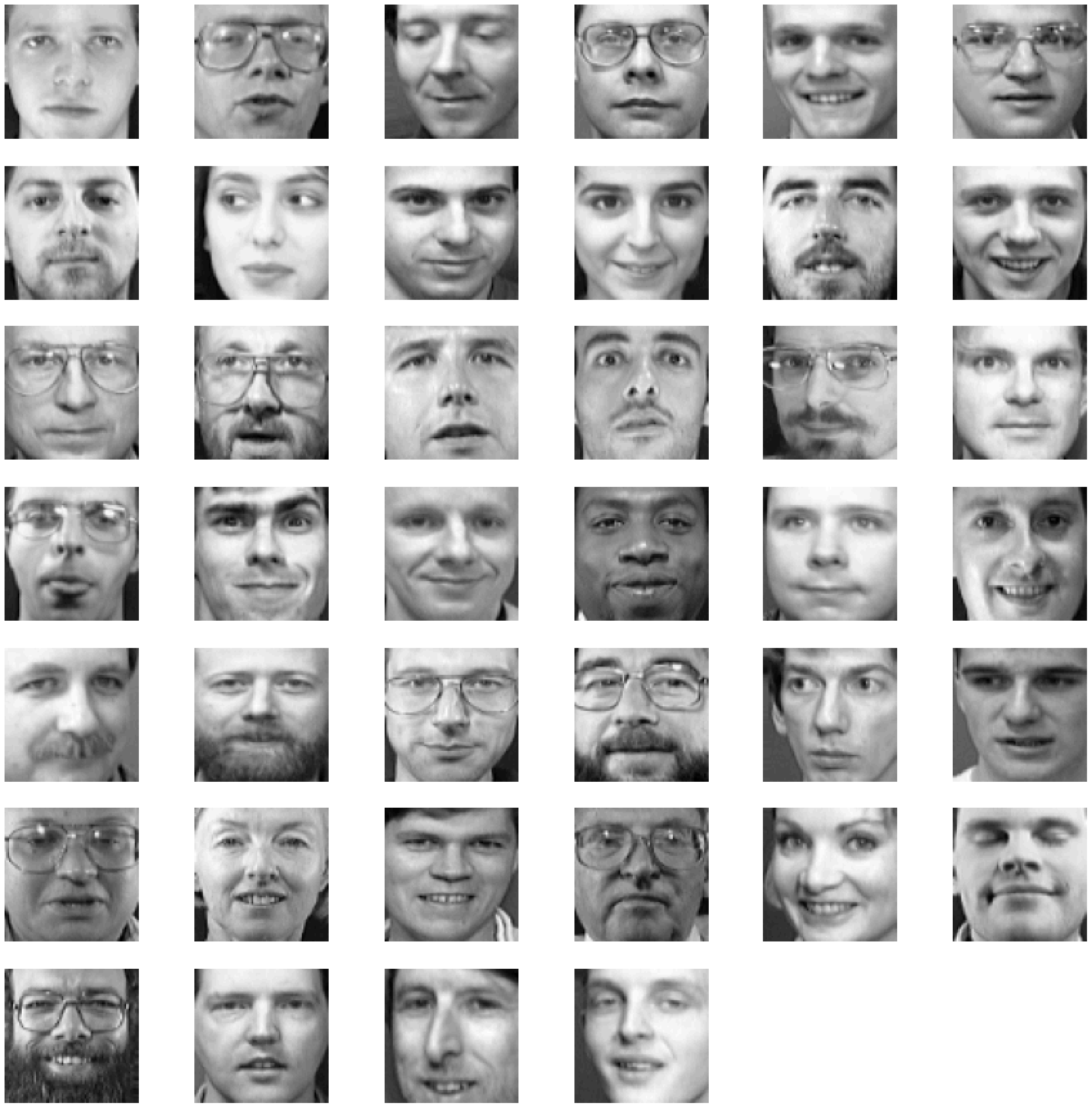
In [9]:
```
faces.keys()
```

Out[9]:
```
dict_keys(['data', 'images', 'target', 'DESCR'])
```

In [4]:
```
X = faces.data # data matrix
t = faces.target # target label

X.shape, t.shape # 400 images, each of size 64x64=4096 pixels
```

```
Out[4]: ((400, 4096), (400,))
```

```
In [10]: fig = plt.figure(figsize=(10,10))
         for i in range(40):
             fig.add_subplot(7,6,i+1)
             idx = np.random.choice(np.where(t==i)[0])
             plt.imshow(X[idx,:].reshape(64,64), cmap='gray')
             plt.axis('off')
```



```
In [11]: np.unique(t, return_counts=True)
```

```
Out[11]: (array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
                 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
                 34, 35, 36, 37, 38, 39]),
          array([10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
                 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
                 10, 10, 10, 10, 10, 10], dtype=int64))
```

In [12]:
```python
X_train, X_test, t_train, t_test = train_test_split(X, t,
                                                    test_size=0.2,
                                                    random_state=42)

X_train.shape, t_train.shape, X_test.shape, t_test.shape
```

Out[12]:
```
((320, 4096), (320,), (80, 4096), (80,))
```

Olivetti images are:

64 × 64 = 4096 features per image

A typical PCA rule is:

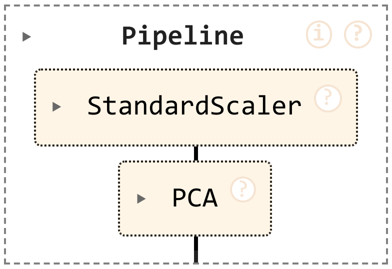
“Keep 5–15% of the original features.”

10% of 4096 ≈ 410

8% of 4096 ≈ 320 - train

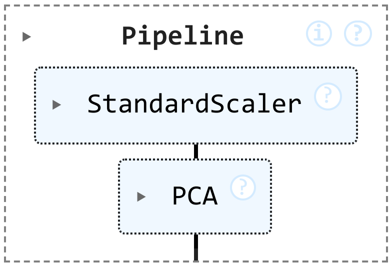
In [13]:
```python
dim_red = Pipeline([('scaler', StandardScaler()),
                   ('pca', PCA(n_components=320))])

dim_red
```

Out[13]:

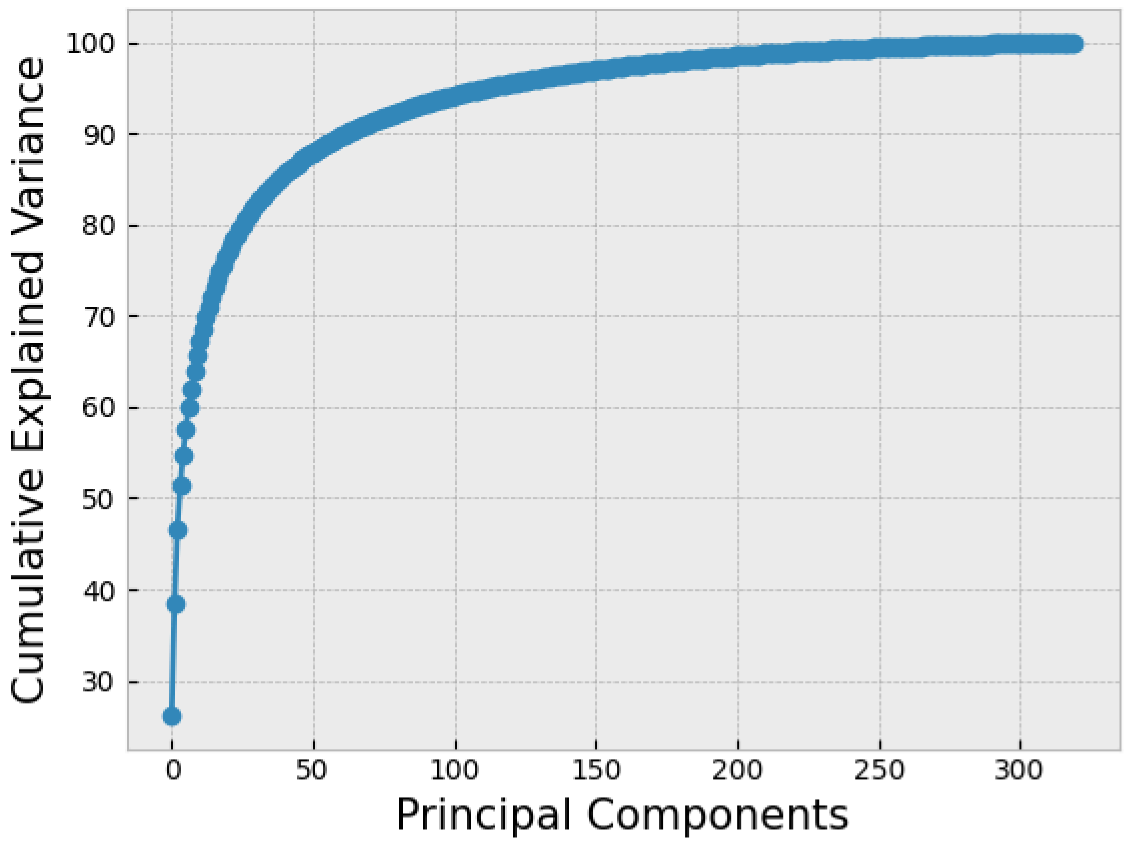


In [14]:
```python
dim_red.fit(X_train)
```

Out[14]:



In [15]:
```python
plt.plot(100*np.cumsum(dim_red.named_steps.pca.explained_variance_ratio_), '-o')
plt.xlabel('Principal Components',size=15)
plt.ylabel('Cumulative Explained Variance', size=15);
```



What the plot shows:

The first ~50 components already capture 80% of the data

Around ~100 components capture 90%

After ~200 components, the curve becomes almost flat → adding more gives very little extra information

```
In [ ]: np.where(np.cumsum(dim_red.named_steps.pca.explained_variance_ratio_)>=0.9)[0] #Thi
```

```
Out[ ]: array([ 62,  63,  64,  65,  66,  67,  68,  69,  70,  71,  72,  73,  74,
                75,  76,  77,  78,  79,  80,  81,  82,  83,  84,  85,  86,  87,
                88,  89,  90,  91,  92,  93,  94,  95,  96,  97,  98,  99, 100,
               101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113,
               114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126,
               127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139,
               140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152,
               153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165,
               166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178,
               179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191,
               192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204,
               205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217,
               218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230,
               231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243,
               244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256,
               257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269,
               270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282,
               283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295,
               296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308,
               309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319], dtype=int64)
```

```
In [20]: np.cumsum(dim_red.named_steps.pca.explained_variance_ratio_)[62]
```

```
Out[20]: 0.900588
```

In order to explain 90% of the variance in the data, we need to preserve 63 principal components.

Let's project to 2-D so we can plot it:

A 2-D scatter plot needs exactly 2 numbers per point

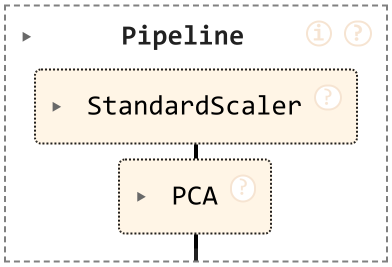
PCA with n_components=2 gives each sample two coordinates:

PC1

PC2

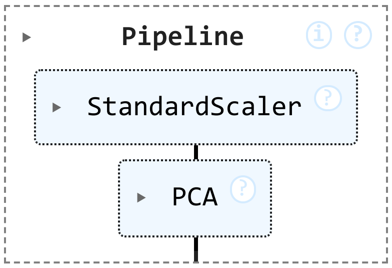
```
In [22]: dim_red_2d = Pipeline([('scaler', StandardScaler()),
                                ('pca', PCA(n_components=2))])

         dim_red_2d
```

Out[22]:



```
In [23]: dim_red_2d.fit(X_train)
```

Out[23]:

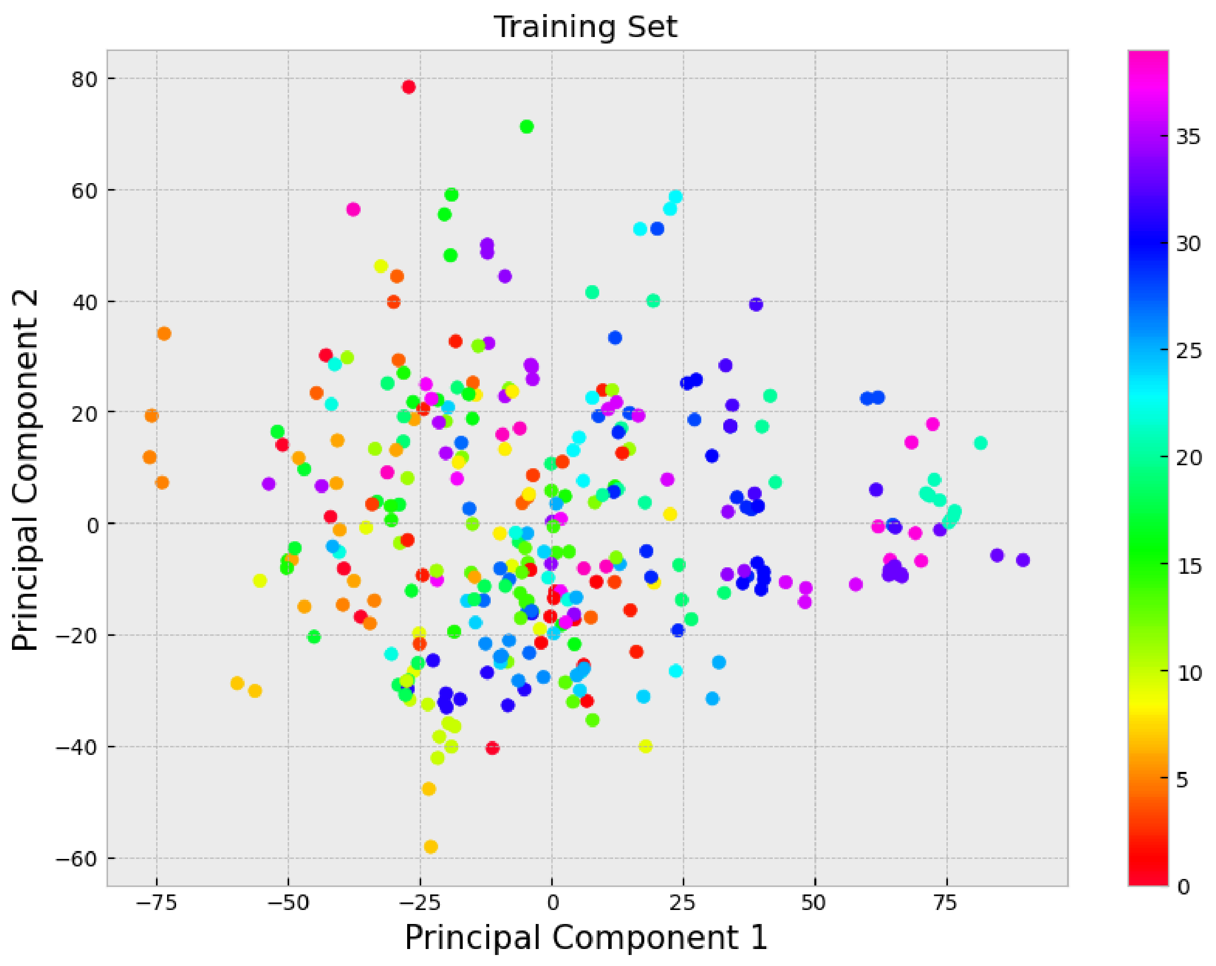


In [25]:

```python
ypca = dim_red_2d.fit_transform(X_train)
ypca
```

```
Out[25]:  array([[-2.70944252e+01,  7.83209686e+01],
                 [ 6.72148705e+00, -3.20074348e+01],
                 [ 3.99888039e+01,  1.73221912e+01],
                 [ 1.29644947e+01, -7.38607073e+00],
                 [-4.52637339e+00, -1.58571501e+01],
                 [-2.60622540e+01,  1.86051941e+01],
                 [ 5.68955183e+00, -2.67705479e+01],
                 [ 3.91179771e+01, -7.18105316e+00],
                 [-1.83818436e+01, -3.65548897e+01],
                 [-7.51024723e+00, -7.72771215e+00],
                 [ 1.68508797e+01,  5.27771645e+01],
                 [ 6.57840424e+01, -9.21890068e+00],
                 [-2.87795010e+01, -3.61557341e+00],
                 [-4.03316078e+01, -5.21210718e+00],
                 [-6.17820644e+00, -3.39243150e+00],
                 [-8.28484058e+00, -3.27676620e+01],
                 [-2.17161827e+01, -1.02336206e+01],
                 [ 6.20750847e+01, -6.58074558e-01],
                 [ 2.36089039e+01,  5.85865974e+01],
                 [-5.96735725e+01, -2.88407478e+01],
                 [-1.96428280e+01,  2.07643871e+01],
                 [-3.34927330e+01,  1.32931404e+01],
                 [ 8.55713081e+00, -1.05650301e+01],
                 [-4.69134178e+01, -1.50423717e+01],
                 [-5.01044312e+01, -6.76714706e+00],
                 [-4.27925835e+01,  3.01035233e+01],
                 [ 6.64231002e-01, -1.23508186e+01],
                 [-4.48983717e+00, -7.12712812e+00],
                 [-3.54754829e+00,  2.58295898e+01],
                 [-8.81691933e+00,  4.43090782e+01],
                 [ 2.65860391e+00, -2.86598682e+01],
                 [ 6.43332901e+01, -6.66046810e+00],
                 [ 2.40636024e+01, -1.93025723e+01],
                 [ 7.02192841e+01, -6.84245968e+00],
                 [ 3.70296211e+01,  2.85286283e+00],
                 [-3.75066986e+01, -1.04025555e+01],
                 [-2.29079666e+01, -5.81707878e+01],
                 [ 8.95105743e+00,  1.91702824e+01],
                 [ 3.39885597e+01,  1.73292332e+01],
                 [ 6.48325424e+01, -3.70923817e-01],
                 [-2.80537491e+01,  2.69348927e+01],
                 [-2.12549114e+01, -3.84011993e+01],
                 [ 4.37904549e+00, -1.73537312e+01],
                 [ 1.61434337e-01,  2.45465115e-01],
                 [ 9.77322102e+00,  2.38481846e+01],
                 [-4.69560165e+01,  9.66736794e+00],
                 [-3.92615819e+00,  2.84225063e+01],
                 [ 6.40771713e+01, -9.38898563e+00],
                 [-2.62637978e+01,  2.17057285e+01],
                 [ 5.43547916e+00, -3.00861740e+01],
                 [-1.21635303e+01, -2.68891258e+01],
                 [-4.93814135e+00, -1.42028761e+01],
                 [-4.60318470e+00, -1.85959351e+00],
                 [-3.31293488e+01,  3.84526682e+00],
                 [-2.73301830e+01,  8.04225159e+00],
                 [-2.81243343e+01,  1.46272402e+01],
[-4.17832603e+01,  2.13252220e+01],
[-1.51377463e+00, -2.76948528e+01],
[ 7.62429428e+01,  1.13162601e+00],
[ 2.57775688e+01,  2.50930424e+01],
[-3.40374908e+01,  3.37372804e+00],
[-3.54485512e+00, -1.59524174e+01],
[ 3.72224464e+01, -9.57060051e+00],
[ 2.62358570e+00,  4.79472351e+00],
[-2.80101852e+01,  1.91297417e+01],
[ 1.21166983e+01,  3.32505569e+01],
[-7.62765961e+01,  1.18098364e+01],
[ 1.47790098e+01,  1.32396135e+01],
[-2.03146019e+01,  5.54173622e+01],
[ 4.83165703e+01, -1.17176447e+01],
[-4.19390182e+01,  1.10219455e+00],
[ 7.38168564e+01, -1.23739064e+00],
[-8.01698208e+00, -1.01516075e+01],
[ 1.20663319e+01,  6.58132601e+00],
[-4.01010180e+00, -8.42817497e+00],
[ 2.42300644e+01, -7.52365112e+00],
[-1.95404282e+01, -3.59667664e+01],
[ 1.94765606e+01, -1.07156706e+01],
[-3.73635793e+00, -1.62402802e+01],
[-3.76302261e+01,  5.63278198e+01],
[-1.79150181e+01,  7.93472862e+00],
[ 1.77807007e+01,  3.66281080e+00],
[ 1.48486176e+01,  1.97543163e+01],
[-1.91592655e+01,  4.80781288e+01],
[-1.19366875e+01,  3.22854424e+01],
[-9.77037048e+00, -1.88265765e+00],
[-9.52153969e+00, -2.40980320e+01],
[ 4.37029457e+00, -2.17759094e+01],
[ 3.35404539e+00, -5.16686106e+00],
[-4.11575928e+01,  2.84965668e+01],
[ 1.79211121e+01, -4.01610718e+01],
[-2.95130272e+01,  1.30896111e+01],
[-8.08006382e+00,  2.41196461e+01],
[-2.61833248e+01, -2.67117424e+01],
[-4.49902058e+00, -1.40015926e+01],
[ 3.43451996e+01,  2.11259174e+01],
[-3.52028580e+01, -8.38351905e-01],
[ 2.25529461e+01,  5.64210472e+01],
[-4.79925232e+01,  1.16720886e+01],
[ 1.34509563e+01,  1.25579996e+01],
[-3.48292184e+00,  8.56902695e+00],
[ 2.09333096e-03,  5.79618025e+00],
[-2.00731697e+01,  1.82573490e+01],
[-1.27054262e+01, -1.13690100e+01],
[-4.92808838e+01, -6.58727217e+00],
[-3.11275387e+01,  2.50798664e+01],
[ 3.39450722e+01,  1.74407177e+01],
[ 7.37328415e+01,  4.06064034e+00],
[-2.24718323e+01, -2.46999569e+01],
[-3.43810655e-02,  1.06295319e+01],
[-8.37953377e+00, -2.49615307e+01],
[ 9.17119086e-01, -5.33539200e+00],
[ 1.74273467e+00, -1.23130140e+01],
[-1.43692894e+01,  2.30266666e+01],
[ 8.95923843e+01, -6.69287395e+00],
[ 3.35188217e+01,  1.98869884e+00],
[ 6.41714783e+01, -8.52599239e+00],
[-4.36355515e+01,  6.63409758e+00],
[ 8.46638641e+01, -5.82564640e+00],
[ 3.63414345e+01, -1.08296986e+01],
[-2.71918869e+01, -2.83062344e+01],
[ 1.32405910e+01,  1.70211067e+01],
[ 2.71863861e+01,  1.85351276e+01],
[ 7.27101669e+01,  7.74631453e+00],
[ 5.99794388e+01,  2.23502827e+01],
[-1.70158577e+01,  1.17815008e+01],
[ 1.20205202e+01, -1.05921936e+01],
[-1.96253860e+00, -2.15226955e+01],
[ 3.88872108e+01,  3.92488747e+01],
[-8.75687027e+00, -1.13294096e+01],
[-5.97455084e-01, -9.84487820e+00],
[ 1.15500956e+01,  2.38861103e+01],
[-5.20148506e+01,  1.63747463e+01],
[ 3.86034012e+01,  5.26815176e+00],
[ 1.80618591e+01, -5.04963779e+00],
[-2.17967091e+01, -8.59567642e+00],
[ 1.07595081e+01,  2.04644547e+01],
[-4.20045471e+00, -2.33654041e+01],
[ 6.91328583e+01, -1.83000231e+00],
[ 2.01257210e+01,  5.28366241e+01],
[ 1.23585405e+01,  2.16791134e+01],
[-2.16191998e+01,  2.20687065e+01],
[ 3.18208714e+01, -2.50369511e+01],
[-5.01994705e+00, -4.50714493e+00],
[-5.02635574e+01, -8.09626007e+00],
[ 3.28414955e+01, -1.25535345e+01],
[-1.79018288e+01,  2.42975330e+01],
[-3.11765728e+01,  9.09470081e+00],
[-3.94036713e+01, -8.19841385e+00],
[ 1.26479225e+01,  6.04681206e+00],
[-3.87699928e+01,  2.96903248e+01],
[ 2.36297245e+01, -2.66236591e+01],
[ 4.13481236e+00,  1.30370436e+01],
[-2.42878723e+01,  2.04529610e+01],
[ 8.24034023e+00,  3.70324183e+00],
[-8.82075596e+00,  2.27560616e+01],
[ 3.08206749e+00, -1.37942009e+01],
[ 1.64994373e+01,  1.92734184e+01],
[-2.27693176e+01,  2.22960358e+01],
[-5.36703377e+01,  7.00619888e+00],
[ 1.79059708e+00,  7.27657676e-01],
[-1.38963213e+01,  3.18122978e+01],
[-5.62979088e+01, -3.01701450e+01],
[ 6.16677361e+01,  5.96487331e+00],
[-2.44445381e+01, -9.41419506e+00],
[-8.83152008e+00,  1.32411432e+01],
[ 6.84205246e+01,  1.44898767e+01],
[-1.89695892e+01, -4.02235222e+01],
[-9.66024303e+00, -2.50807705e+01],
[ 3.80083885e+01,  2.42992592e+00],
[-1.52375603e+01, -8.95928192e+00],
[-1.40040982e+00, -5.17880774e+00],
[-2.77080173e+01, -3.01268139e+01],
[ 1.89091339e+01, -9.71826458e+00],
[-2.93455639e+01,  4.43039665e+01],
[ 5.78055801e+01, -1.10578442e+01],
[-4.01984253e+01, -1.26158667e+00],
[-7.42494154e+00,  2.36302853e+01],
[-1.59805470e+01, -1.39958487e+01],
[-2.53465538e+01, -2.51610756e+01],
[-3.03820076e+01,  4.96953577e-01],
[-1.73037335e-01, -1.68208332e+01],
[-2.99635239e+01,  3.97115631e+01],
[-2.73147755e+01, -3.07739854e+00],
[-4.50723534e+01, -2.04843864e+01],
[-5.93902349e+00, -1.25737085e+01],
[-2.03845062e+01, -3.22200012e+01],
[-1.55864534e+01,  2.55703664e+00],
[-3.03849468e+01, -2.35874786e+01],
[-2.50922966e+01, -1.98567734e+01],
[-3.36144638e+01, -1.39422789e+01],
[-1.44234123e+01, -1.79201584e+01],
[ 1.27058477e+01,  1.62490616e+01],
[ 3.98182030e+01, -1.19638309e+01],
[-4.87304344e+01, -4.53356504e+00],
[ 7.48856115e+00, -1.69768085e+01],
[-1.73552475e+01, -3.16921329e+01],
[ 7.75214863e+00,  4.14276924e+01],
[ 3.05778160e+01, -3.15920124e+01],
[ 6.64834824e+01, -9.20592213e+00],
[-3.61927757e+01, -1.68813553e+01],
[-4.15438538e+01, -4.19895697e+00],
[-2.00465469e+01,  1.25532885e+01],
[-2.34669590e+01, -3.25982513e+01],
[-6.78606224e+00, -1.74250793e+00],
[ 3.66298447e+01, -8.62183094e+00],
[ 7.11798325e+01,  5.31295109e+00],
[-2.65583763e+01, -1.22017937e+01],
[-3.23921471e+01,  4.61391373e+01],
[ 8.15314407e+01,  1.43474646e+01],
[-1.45704193e+01, -9.73146057e+00],
[-1.49067278e+01,  2.51888714e+01],
[ 2.10063124e+00,  1.09784937e+01],
[-9.75616169e+00, -2.40244961e+01],
[ 4.25469894e+01,  7.30445910e+00],
[-3.67717314e+00,  2.80260239e+01],
[-1.81809521e+01,  3.26157150e+01],
[-5.13119888e+00, -2.99135685e+01],
[ 6.05547571e+00,  7.56354570e+00],
[-8.05734444e+00, -2.11138783e+01],
[ 6.20258904e+01,  2.25440044e+01],
[ 1.98599601e+00, -1.83198395e+01],
[-2.38406525e+01,  2.48988113e+01],
[ 7.82782316e+00, -3.54097290e+01],
[-4.08496056e+01,  7.11774445e+00],
[ 4.07781410e+00, -3.21602440e+01],
[-1.49972620e+01, -2.02276468e-01],
[-5.10442162e+01,  1.40279274e+01],
[-4.06440430e+01,  1.48340540e+01],
[ 2.75063457e+01,  2.57366085e+01],
[ 4.51272279e-01, -1.35308008e+01],
[-1.84493599e+01, -1.95404568e+01],
[-5.55000114e+00,  3.54890275e+00],
[ 2.20095730e+01,  7.78562832e+00],
[ 6.53333054e+01, -8.14880908e-01],
[ 7.65957031e+01,  2.09645605e+00],
[-1.71279240e+01,  1.44149284e+01],
[ 4.29764748e+00, -1.63960991e+01],
[ 5.25633669e+00,  1.53656034e+01],
[ 1.93449078e+01,  3.99315948e+01],
[ 7.58891144e+01,  6.63163662e-01],
[ 1.18328276e+01,  5.57026768e+00],
[ 4.81627655e+01, -1.42417336e+01],
[ 8.97675037e-01,  3.45348167e+00],
[ 6.15783453e+00, -8.16087627e+00],
[ 4.45025864e+01, -1.06317158e+01],
[ 4.03271408e+01, -1.01651011e+01],
[-1.25103016e+01, -2.16812744e+01],
[ 4.15636253e+01,  2.28498611e+01],
[-1.49899168e+01,  1.87397976e+01],
[-4.51010036e+00,  4.62093115e+00],
[ 7.54480515e+01,  8.98470655e-02],
[ 2.66007690e+01, -1.73119392e+01],
[ 4.03043137e+01, -8.80665398e+00],
[ 4.81842422e+00, -2.73745823e+01],
[-2.68888779e+01, -3.17783051e+01],
[-7.39386749e+01,  7.25482893e+00],
[-1.12070055e+01, -4.04931602e+01],
[-4.46167564e+01,  2.33696175e+01],
[ 2.25637245e+01,  1.54552567e+00],
[ 9.71830463e+00,  4.99930096e+00],
[-2.50217209e+01, -2.17337379e+01],
[-9.40853405e+00, -2.38230553e+01],
[-2.21284986e+00, -1.90645485e+01],
[ 6.52282486e+01, -7.75596380e+00],
[-3.95877380e+01, -1.47135763e+01],
[-3.87288071e-02, -7.34728193e+00],
[-6.24166059e+00, -2.83487587e+01],
[ 1.75089169e+01, -3.12083626e+01],
[ 6.13264561e+00, -2.55355682e+01],
[-1.99207249e+01, -3.31261787e+01],
[-4.24512053e+00,  5.13747692e+00],
[-1.28838081e+01, -1.39176044e+01],
[-1.89445343e+01,  5.89736824e+01],
[ 3.31038742e+01,  2.82930851e+01],
[-5.98744011e+00,  1.69951324e+01],
[-5.59522867e+00, -8.88614178e+00],
[ 3.92783318e+01,  3.03626871e+00],
[ 7.24151001e+01,  1.77653885e+01],
[-2.00247040e+01, -3.06372623e+01],
       [ 4.42759633e-01, -1.98651943e+01],
       [-7.59048615e+01,  1.92641373e+01],
       [-3.44696922e+01, -1.80532875e+01],
       [-2.89236012e+01,  3.30213833e+00],
       [-1.46460152e+01, -1.37190504e+01],
       [ 1.04161968e+01, -7.78379822e+00],
       [-2.90952568e+01, -2.91381359e+01],
       [-4.69759321e+00,  7.12236633e+01],
       [-7.35523376e+01,  3.40253944e+01],
       [-2.90561638e+01,  2.92637558e+01],
       [-1.21730061e+01,  4.99735146e+01],
       [ 7.78255844e+00,  2.24825897e+01],
       [ 4.71040249e+00, -1.33770084e+01],
       [ 1.61326733e+01, -2.31739635e+01],
       [-2.72579803e+01, -2.98429050e+01],
       [-1.57026100e+01,  2.31603355e+01],
       [-3.75619364e+00, -1.57833948e+01],
       [ 2.47455387e+01, -1.38177843e+01],
       [ 3.52412758e+01,  4.56925440e+00],
       [ 3.34178238e+01, -9.26124859e+00],
       [ 6.18852282e+00, -2.61837406e+01],
       [ 3.05029907e+01,  1.20641499e+01],
       [ 3.64156872e-01, -6.44995570e-01],
       [-5.84534168e+00, -1.71213226e+01],
       [-3.05059071e+01,  3.04411650e+00],
       [-2.13330116e+01,  1.80499039e+01],
       [-5.53503113e+01, -1.03658838e+01],
       [ 2.73598766e+00, -1.78632698e+01],
       [-1.76610336e+01,  1.08901510e+01],
       [ 6.65900879e+01, -9.66872597e+00],
       [ 7.18127670e+01,  4.84204245e+00],
       [ 1.22980566e+01, -6.25303030e+00],
       [-9.26713943e+00,  1.58840580e+01],
       [ 1.50026960e+01, -1.56618881e+01],
       [-2.77151051e+01, -3.08764420e+01],
       [-2.32878952e+01, -4.78028831e+01],
       [-2.15886116e+01, -4.22427101e+01],
       [-9.69488430e+00, -8.19389153e+00],
       [-1.21369753e+01,  4.85725555e+01],
       [-2.74754372e+01, -2.83948021e+01]], dtype=float32)
```

In [12]:
```
plt.figure(figsize=(10,7))
plt.scatter(ypca[:,0], ypca[:,1], c=t_train, cmap=plt.cm.gist_rainbow)
plt.xlabel('Principal Component 1', size=15)
plt.ylabel('Principal Component 2', size=15)
plt.title('Training Set')
plt.colorbar();
```



Not that the 40 classes are overlapping in the linear projection space. This is because PCA is **unsupervised**, it does use the class labels *anywhere* in finding the matrix for linear projection.

To apply this transformation in the test set, simply multiply the resultant modal matrix with the scaled test set:

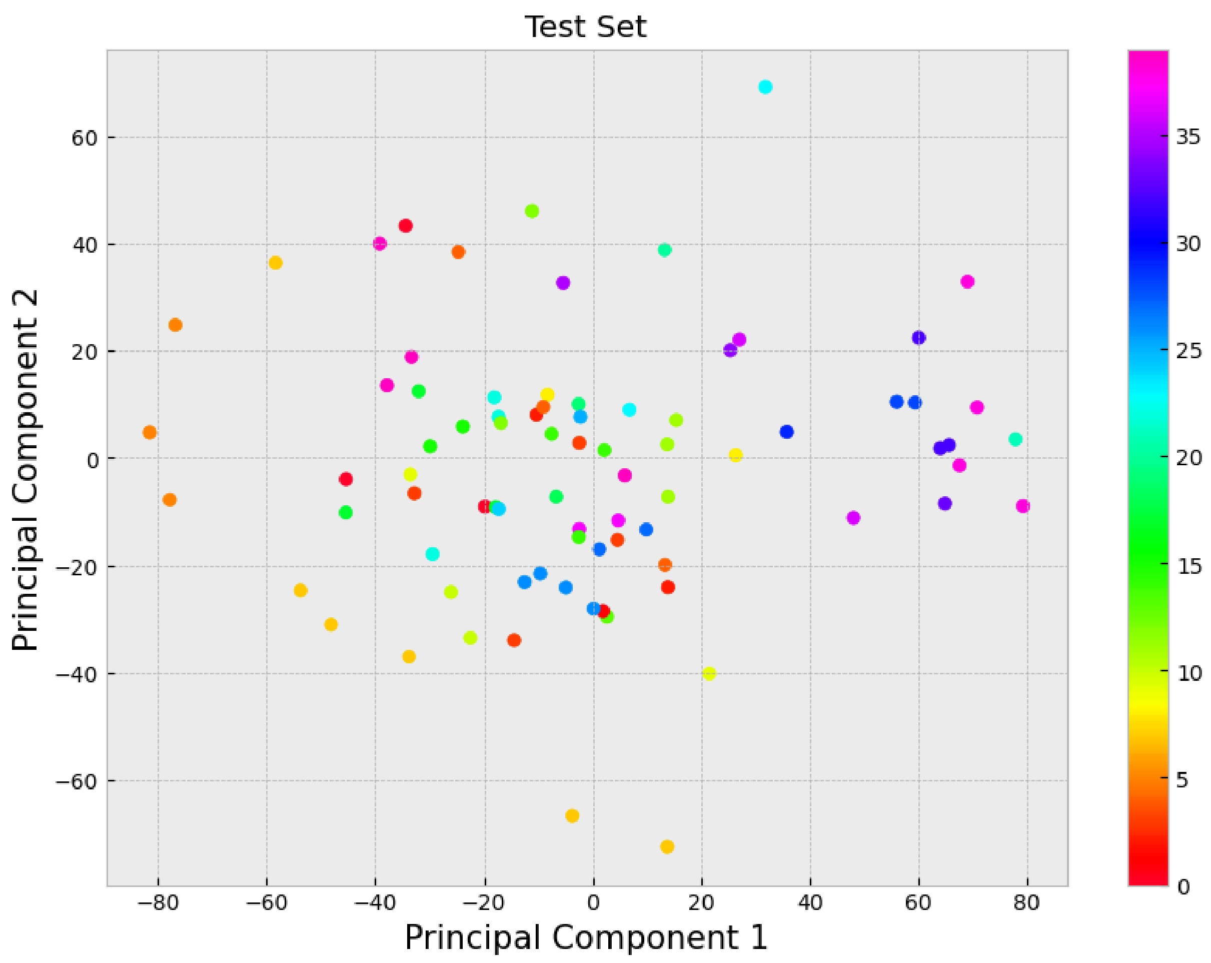
In [26]:
```
# Transform the test set using the linear transformation found with the training da
ypca_test = dim_red_2d.transform(X_test)

ypca_test.shape
```

Out[26]:
```
(80, 2)
```

In [27]:
```
ypca_test
```

```
Out[27]: array([[ 13.2673435 ,  38.752693  ],
                [ 59.382866  ,  10.292743  ],
                [-32.801197  ,  -6.606206  ],
                [ 77.86476   ,   3.4806166 ],
                [ 21.509811  , -40.16344   ],
                [ -8.268763  ,  11.77495   ],
                [ 64.00316   ,   1.7863787 ],
                [-33.54249   ,  -3.0781236 ],
                [ -9.626152  , -21.551834  ],
                [-11.12121   ,  45.967407  ],
                [-19.787346  ,  -9.077597  ],
                [ 48.030052  , -11.163881  ],
                [-76.81553   ,  24.743196  ],
                [-48.129852  , -31.04309   ],
                [  2.6499858 , -29.568161  ],
                [ 13.328286  , -19.91267   ],
                [  9.894091  , -13.362189  ],
                [ -2.4443336 , -13.2623005 ],
                [  6.804084  ,   8.983894  ],
                [ 70.80363   ,   9.413528  ],
                [-53.763264  , -24.699272  ],
                [  1.9308505 , -28.521421  ],
                [-33.323677  ,  18.783146  ],
                [  1.2481086 , -17.005035  ],
                [-45.370163  ,  -3.9758565 ],
                [-39.185486  ,  39.897102  ],
                [ 15.419554  ,   7.024021  ],
                [-18.102127  ,  11.26958   ],
                [-12.522806  , -23.115448  ],
                [-22.471298  , -33.531097  ],
                [  5.941933  ,  -3.267729  ],
                [ -2.5507503 ,  10.011581  ],
                [ -4.9104767 , -24.152012  ],
                [-77.81635   ,  -7.8069105 ],
                [ 31.814444  ,  69.07559   ],
                [ 13.726966  ,   2.5200431 ],
                [ 13.885804  ,  -7.258341  ],
                [ 25.349064  ,  20.04554   ],
                [-23.887115  ,   5.8399315 ],
                [ -7.5245905 ,   4.4517946 ],
                [ 79.265526  ,  -8.964696  ],
                [-81.50955   ,   4.727819  ],
                [ -3.7334778 , -66.64203   ],
                [-10.345794  ,   8.023443  ],
                [ 26.3612    ,   0.5275583 ],
                [ 69.02699   ,  32.83768   ],
                [  2.163819  ,   1.4406579 ],
                [ -6.7082376 ,  -7.2545347 ],
                [ 13.863986  , -24.094048  ],
                [-17.836912  ,  -9.186252  ],
                [ -9.074993  ,   9.485323  ],
                [ 65.62421   ,   2.38415   ],
                [ 64.87576   ,  -8.4946785 ],
                [-33.786102  , -37.023182  ],
                [  4.7214713 , -11.650687  ],
                [-14.433821  , -33.97001   ],
                [-17.302525  ,   7.643322  ],
                [-31.99989   ,  12.4264555 ],
                [ -2.427134  ,   2.8031218 ],
                [-29.904675  ,   2.1460943 ],
                [-16.851395  ,   6.500622  ],
                [ 35.76239   ,   4.837394  ],
                [ -2.2127972 ,   7.634941  ],
                [-58.35209   ,  36.322906  ],
                [-26.055082  , -25.00381   ],
                [  4.594894  , -15.276977  ],
                [ -5.391798  ,  32.622063  ],
                [  0.19394422, -28.073814  ],
                [-37.83949   ,  13.5049515 ],
                [ 13.74098   , -72.41014   ],
                [ 60.04293   ,  22.356941  ],
                [ -2.518438  , -14.735438  ],
                [-34.396084  ,  43.229065  ],
                [-24.713623  ,  38.36746   ],
                [ 67.57144   ,  -1.4050589 ],
                [-17.258284  ,  -9.521631  ],
                [-29.463581  , -17.924345  ],
                [ 27.044024  ,  22.016058  ],
                [-45.405014  , -10.1932535 ],
                [ 55.999     ,  10.441351  ]], dtype=float32)
```

In [14]:
```
plt.figure(figsize=(10,7))
plt.scatter(ypca_test[:,0], ypca_test[:,1], c=t_test, cmap=plt.cm.gist_rainbow)
plt.xlabel('Principal Component 1', size=15)
plt.ylabel('Principal Component 2', size=15)
plt.title('Test Set')
plt.colorbar();
```



You can access the linear transformation $\mathbf{A} = \mathbf{U}^T$ using the method `components_`:

In [28]:
```
A = dim_red_2d.named_steps.pca.components_

A.shape
```

Out[28]:
```
(2, 4096)
```

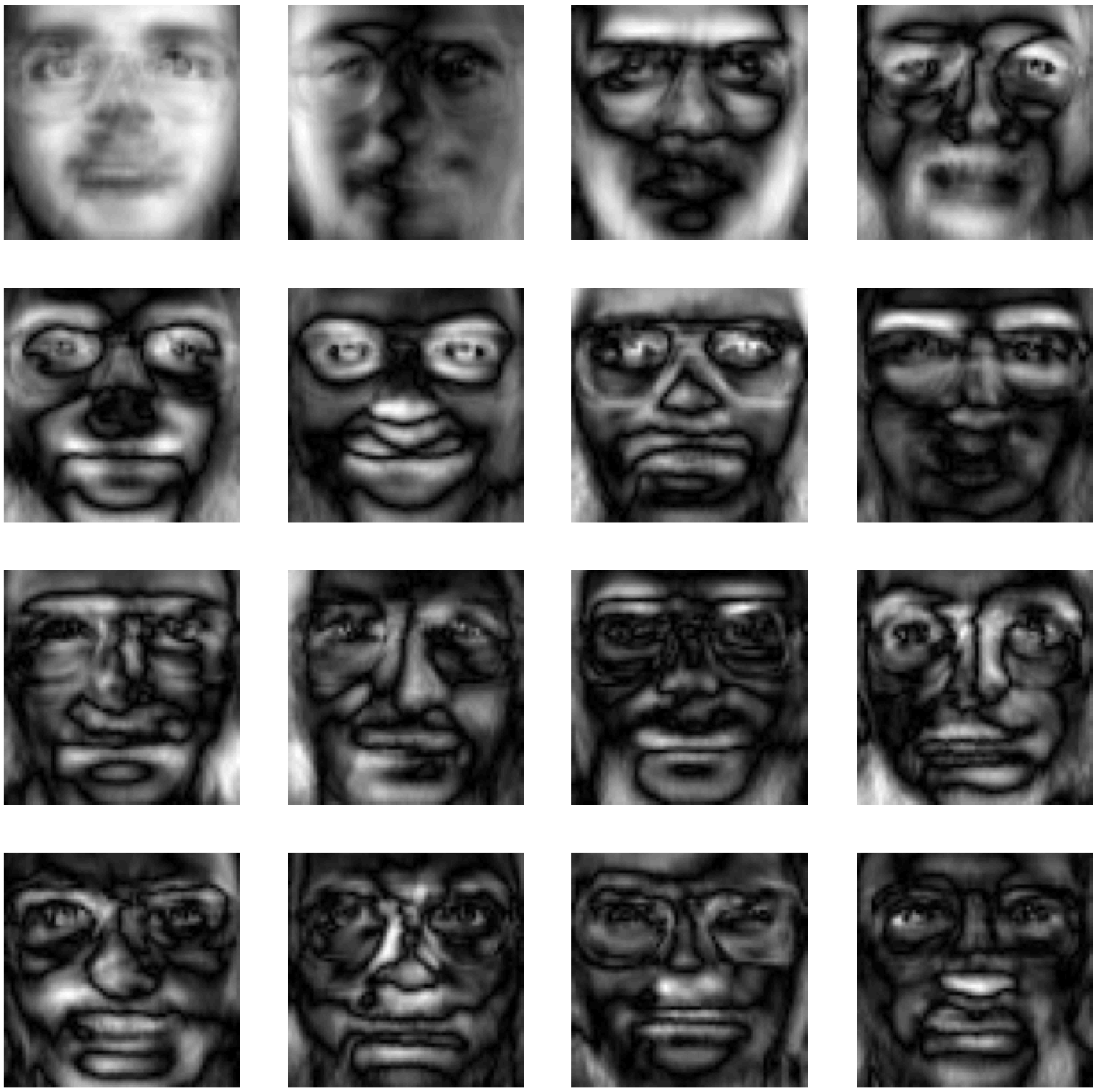
Note that the eigenvectors are described in the original space, that is, they are 4096-dimensional!

Since we are working with images, we can reshape them back to a 64×64 image and see what are the regions in the image with maximum explained variance! This is called the **eigenfaces**.

Let's now recover 16 eigenvectors and plot them as images:

In [29]:
```
n_components = 16

# Dimensionality reduction pipeline
dim_red = Pipeline([('scaler', StandardScaler()),
                   ('pca', PCA(n_components=n_components))])
dim_red.fit(X_train)
```

```
# Linear projections
ypca = dim_red.fit_transform(X_train)

# Visualization of eigenvectors/the new features/principal components
fig=plt.figure(figsize=(10,10))
for i in range(n_components):
    fig.add_subplot(4,4,i+1)
    plt.imshow(abs(dim_red.named_steps.pca.components_[i,:].reshape(64,64)),cmap='g
    plt.axis('off')
```



The eigenvectors are describing the regions in the 64x64 image that explain the most variance. the more eigenvectors are kept, the better a reconstruction image will be produced.

Since the projection is given by:

$$\mathbf{Y} = \mathbf{A}\mathbf{X}$$

In order to recover $\mathbf{X}$, we need to left-multiply by the pseudo-inverse of $\mathbf{A}$:

$$\hat{\mathbf{X}} = \mathbf{A}^{\dagger}\mathbf{Y}$$

In `scikit-learn`, this step can be computed by using the method `inverse_transform`.

You took your original image (X) → applied PCA → got compressed data (Y).

This is written as:

Y = A X

X = original image (4096 pixels)

A = PCA transformation matrix

Y = compressed version (e.g., 100 PCA components)

Now the question is:

👉 How do we go back from Y to X? (How do we reconstruct the image?)

To reverse the transformation, we need something like:

X ≈ A^+ Y

Where A^+ is the pseudo-inverse of A (because A is not square).

fittransfrom above

inverse transform below

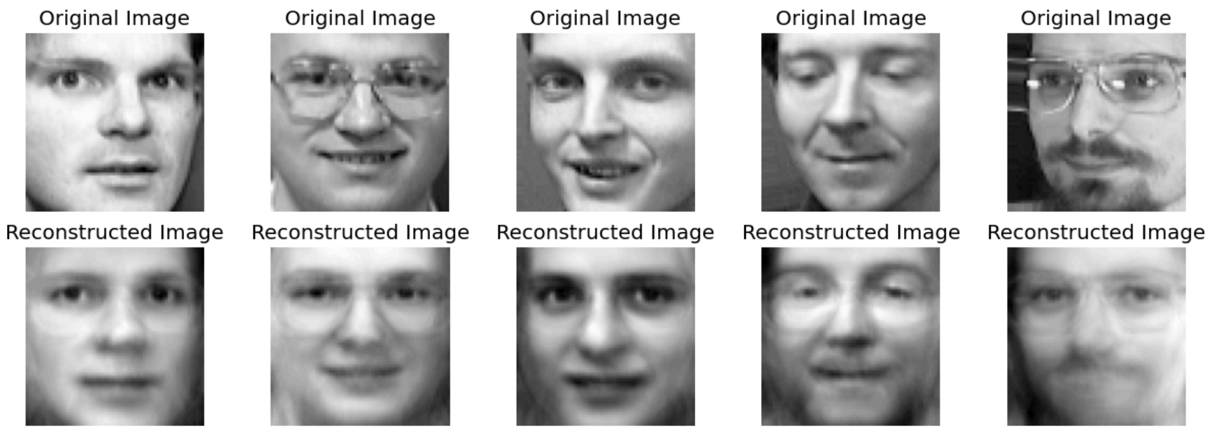
```
In [30]: ypca.shape
```

```
Out[30]: (320, 16)
```

```
In [31]: # Reconstruction
X_reconst = dim_red.inverse_transform(ypca)

X_reconst.shape
```

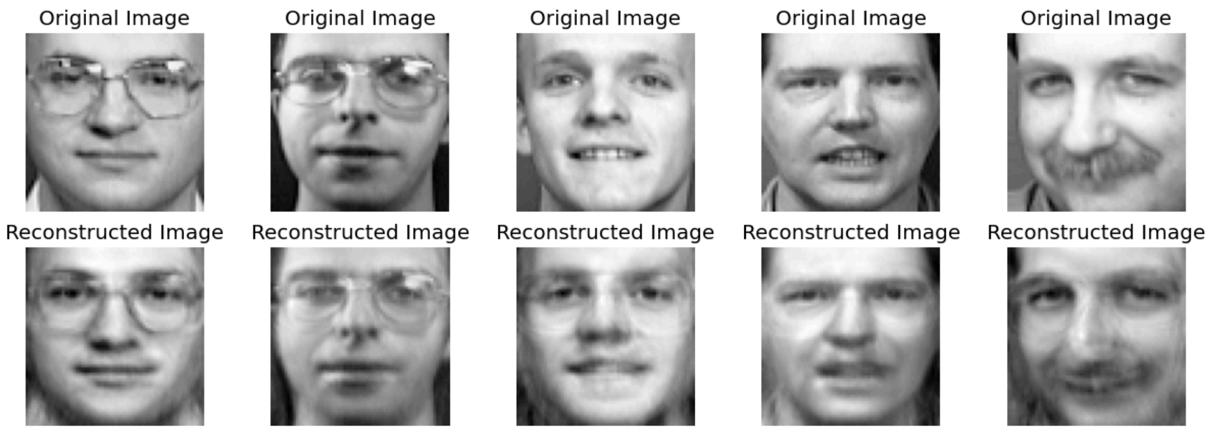
```
Out[31]: (320, 4096)
```

```
In [36]: N = 5
idx = np.random.choice(range(X_reconst.shape[0]),replace=False,size=N)

fig = plt.figure(figsize=(15,5))

j=1
for i in range(N):
    fig.add_subplot(2,N,j)
    plt.imshow(X_train[idx[i],:].reshape(64,64), cmap='gray')
    plt.axis('off')
    plt.title('Original Image');

    fig.add_subplot(2,N,j+N)
    plt.imshow(X_reconst[idx[i],:].reshape(64,64), cmap='gray')
    plt.axis('off')
    plt.title('Reconstructed Image');
    j+=1
```



Putting it all together:

In [38]:
```
n_components = 63

# Dimensionality reduction pipeline
dim_red = Pipeline([('scaler', StandardScaler()),
                   ('pca', PCA(n_components=n_components))])
dim_red.fit(X_train)

# Linear projections
ypca = dim_red.fit_transform(X_train)

# Reconstruction
X_reconst = dim_red.inverse_transform(ypca)

# Visualizations
N = 5 # number of images to display
fig = plt.figure(figsize=(15,5))
idx = np.random.choice(range(X_reconst.shape[0]),replace=False,size=N)
j=1
for i in range(N):
    fig.add_subplot(2,N,j)
    plt.imshow(X_train[idx[i],:].reshape(64,64), cmap='gray')
    plt.axis('off')
    plt.title('Original Image');

    fig.add_subplot(2,N,j+N)
    plt.imshow(X_reconst[idx[i],:].reshape(64,64), cmap='gray')
    plt.axis('off')
    plt.title('Reconstructed Image');
    j+=1
```



PCA as a preprocessing step

In [39]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score, KFold, StratifiedKFold
import time
```

In [40]:
```
# Model without Dimensionality reduction
model = Pipeline([('scaler', StandardScaler()),
                  ('clf', LogisticRegression())])
start = time.time()
model.fit(X_train, t_train)
end = time.time()
print('Training time: ', end-start, ' ms')
```

```
Training time:  0.6019587516784668  ms
```

In [41]:
```
model_dim_red = Pipeline([('scaler', StandardScaler()),
                          ('pca', PCA(n_components=0.9)),
                          ('clf', LogisticRegression())])
start = time.time()
model_dim_red.fit(X_train, t_train)
end = time.time()
print('Training time: ', end-start, ' ms')
```

```
Training time:  0.2179427146911621  ms
```

In [42]:
```
model.score(X_test, t_test)
```

Out[42]:
```
0.975
```

In [43]:
```
model_dim_red.score(X_test, t_test)
```

Out[43]:
```
0.975
```

Supervised Linear Dimensionality Reduction via FLDA

FLDA: Fisher's Linear Discriminant Analysis (or LDA)

A very popular type of a linear discriminant is the **Fisher's Linear Discriminant**.

- Given two classes, we can compute the mean of each class:

$$\vec{\mathbf{m}}_1 = \frac{1}{N_1}\sum_{n \in C_1} \vec{\mathbf{x}}_{\mathbf{n}}$$
$$\vec{\mathbf{m}}_{\mathbf{2}} = \frac{1}{N_2}\sum_{n \in C_2} \vec{\mathbf{x}}_{\mathbf{n}}$$

We can maximize the separation of the means:

$$m_2 - m_1 = \vec{\mathbf{w}}^T(\vec{\mathbf{m}}_2 - \vec{\mathbf{m}}_1)$$

- $\vec{\mathbf{w}}^T\vec{\mathbf{x}}$ takes a D dimensional data point and projects it down to 1-D with a weight sum of the original features. We want to find a weighting that maximizes the separation of the class means.

- Not only do we want well separated means for each class, but we also want each class to be *compact* to minimize overlap between the classes.
- Consider the *within class variance:*

$$\begin{aligned} s_k^2 = \sum_{n \in C_k}(y_n - m_k)^2 &= \sum_{n \in C_k} (\vec{\mathbf{w}}^T\vec{\mathbf{x}}_n - m_k)^2 \\ &= \vec{\mathbf{w}}^T\sum_{n \in C_k} (\vec{\mathbf{x}}_n - \vec{\mathbf{m}}_{\mathbf{k}})(\vec{\mathbf{x}}_n - \vec{\mathbf{m}}_{\mathbf{k}})^T\vec{\mathbf{w}} \end{aligned}$$

- So, we want to minimize within class variance and maximize between class separability. How about the following objective function:

$$
\begin{aligned}
J(\mathbf{w}) &= \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} \\
&= \frac{\vec{\mathbf{w}}^T(\vec{\mathbf{m}}_2 - \vec{\mathbf{m}}_1)(\vec{\mathbf{m}}_2 - \vec{\mathbf{m}}_1)^T\vec{\mathbf{w}}}{\sum_{n\in C_1}(\vec{\mathbf{w}}^T\vec{\mathbf{x}}_n - m_1)^2 + \sum_{n\in C_2}(\vec{\mathbf{w}}^T\vec{\mathbf{x}}_n - m_2)^2} \\
&= \frac{\vec{\mathbf{w}}^T(\vec{\mathbf{m}}_2 - \vec{\mathbf{m}}_1)(\vec{\mathbf{m}}_2 - \vec{\mathbf{m}}_1)^T\vec{\mathbf{w}}}{\vec{\mathbf{w}}^T\left(\sum_{n\in C_1}(\vec{\mathbf{x}}_n - \vec{\mathbf{m}}_1)(\vec{\mathbf{x}}_n - \vec{\mathbf{m}}_1)^T + \sum_{n\in C_2}(\vec{\mathbf{x}}_n - \vec{\mathbf{m}}_2)(\vec{\mathbf{x}}_n - \vec{\mathbf{m}}_2)^T\right)\vec{\mathbf{w}}} \\
&= \frac{\vec{\mathbf{w}}^T\mathbf{S}_B\vec{\mathbf{w}}}{\vec{\mathbf{w}}^T\mathbf{S}_W\vec{\mathbf{w}}}
\end{aligned}
$$

where

$$S_B = (\vec{\mathbf{m}}_2 - \vec{\mathbf{m}}_1)(\vec{\mathbf{m}}_2 - \vec{\mathbf{m}}_1)^T$$

and

$$S_W = \frac{1}{N_1}\sum_{n\in C_1}(\vec{\mathbf{x}}_n - \vec{\mathbf{m}}_1)(\vec{\mathbf{x}}_n - \vec{\mathbf{m}}_1)^T + \frac{1}{N_2}\sum_{n\in C_2}(\vec{\mathbf{x}}_n - \vec{\mathbf{m}}_2)(\vec{\mathbf{x}}_n - \vec{\mathbf{m}}_2)^T$$

- Ok, so let's optimize:

$$
\begin{aligned}
\frac{\partial J(\vec{\mathbf{w}})}{\partial \vec{\mathbf{w}}} = \frac{2(\vec{\mathbf{w}}^T\mathbf{S}_W\vec{\mathbf{w}})\mathbf{S}_B\vec{\mathbf{w}} - 2(\vec{\mathbf{w}}^T\mathbf{S}_B\vec{\mathbf{w}})\mathbf{S}_W\vec{\mathbf{w}}}{(\vec{\mathbf{w}}^T\mathbf{S}_W\vec{\mathbf{w}})^2} &= 0 \\
0 &= \frac{\mathbf{S}_B\vec{\mathbf{w}}}{(\vec{\mathbf{w}}^T\mathbf{S}_W\vec{\mathbf{w}})} - \frac{(\vec{\mathbf{w}}^T\mathbf{S}_B\vec{\mathbf{w}})\mathbf{S}_W\vec{\mathbf{w}}}{(\vec{\mathbf{w}}^T\mathbf{S}_W\vec{\mathbf{w}})^2} \\
(\vec{\mathbf{w}}^T\mathbf{S}_W\vec{\mathbf{w}})\mathbf{S}_B\vec{\mathbf{w}} &= (\vec{\mathbf{w}}^T\mathbf{S}_B\vec{\mathbf{w}})\mathbf{S}_W\vec{\mathbf{w}} \\
\mathbf{S}_B\vec{\mathbf{w}} &= \frac{\vec{\mathbf{w}}^T\mathbf{S}_B\vec{\mathbf{w}}}{\vec{\mathbf{w}}^T\mathbf{S}_W\vec{\mathbf{w}}}\mathbf{S}_W\vec{\mathbf{w}} \\
\mathbf{S}_W^{-1}\mathbf{S}_B\vec{\mathbf{w}} &= \lambda\vec{\mathbf{w}}
\end{aligned}
$$

where the scalar $\lambda = \frac{\vec{\mathbf{w}}^T\mathbf{S}_B\vec{\mathbf{w}}}{\vec{\mathbf{w}}^T\mathbf{S}_W\vec{\mathbf{w}}}$

Does this look familiar?

This is the generalized eigenvalue problem!

- So the direction of projection correspond to the eigenvectors of $\mathbf{S}_W^{-1}\mathbf{S}_B$ with the largest eigenvalues.

The solution is easy when $S_w^{-1} = (\Sigma_1 + \Sigma_2)^{-1}$ exists.

In this case, if we use the definition of $S_B = (\vec{\mathbf{m}}_2 - \vec{\mathbf{m}}_1)(\vec{\mathbf{m}}_2 - \vec{\mathbf{m}}_1)^T$:

$$
\begin{aligned}
S_W^{-1} S_B \vec{\mathbf{w}} &= \lambda \vec{\mathbf{w}} \\
S_W^{-1}(\vec{\mathbf{m}}_2 - \vec{\mathbf{m}}_1)(\vec{\mathbf{m}}_2 - \vec{\mathbf{m}}_1)^T \vec{\mathbf{w}} &= \lambda \vec{\mathbf{w}}
\end{aligned}
$$

Noting that $\alpha = (\vec{\mathbf{m}}_2 - \vec{\mathbf{m}}_1)^T \vec{\mathbf{w}}$ is a constant, this can be written as:

$$S_W^{-1}(\vec{\mathbf{m}}_2 - \vec{\mathbf{m}}_1) = \frac{\lambda}{\alpha} \vec{\mathbf{w}}$$

- Since we don't care about the magnitude of $\vec{\mathbf{w}}$:

$$\vec{\mathbf{w}}^* = S_W^{-1}(\vec{\mathbf{m}}_2 - \vec{\mathbf{m}}_1) = (\Sigma_1 + \Sigma_2)^{-1}(\vec{\mathbf{m}}_2 - \vec{\mathbf{m}}_1)$$

Make sure $\vec{\mathbf{w}}^*$ is a unit vector by taking: $\vec{\mathbf{w}}^* \leftarrow \frac{\vec{\mathbf{w}}^*}{\|\vec{\mathbf{w}}^*\|}$

- Note that if the within-class covariance, S_W, is isotropic, so that S_W is proportional to the unit matrix, we find that $\vec{\mathbf{w}}$ is proportional to the difference of the class means.
- This result is known as *Fisher's linear discriminant*, although strictly it is not a discriminant but rather a specific choice of direction for projection of the data down to one dimension. However, the projected data can subsequently be used to construct a discriminant, by choosing a threshold y_0 so that we classify a new point as belonging to C_1 if $y(x) \geq y_0$ and classify it as belonging to C_2 otherwise.

Also, note that:

- For a classification problem with Gaussian classes of equal covariance $\Sigma_i = \Sigma$, the boundary is the plane of normal:

$$\vec{\mathbf{w}} = \Sigma^{-1}(\vec{\mathbf{m}}_i - \vec{\mathbf{m}}_j)$$

- If $\Sigma_2 = \Sigma_1$, this is also the LDA solution.

This gives two different **interpretations** of LDA:

- It is optimal *if and only if* the classes are Gaussian and have equal covariance.

- A classifier on the LDA features, is equivalent to the boundary after the approximation of the data by two Gaussians with equal covariance.

The final discriminant decision boundary is $\vec{\mathbf{y}} = \vec{\mathbf{w}}^* \vec{\mathbf{x}} + w_0$

The *bias* term w_0 can be defined as:

$$w_0 = \frac{1}{2}\left(\frac{1}{N_1}\sum_{n \in C_1} \vec{x}_n + \frac{1}{N_2}\sum_{n \in C_2} \vec{x}_n\right)^T \vec{\mathbf{w}}^*$$

- An extension to multi-class problems has a similar derivation.

Limitations of LDA:

1. LDA produces at most $C - 1$ feature projections, where C is the number of classes.
2. If the classification error estimates establish that more features are needed, some other method must be employed to provide those additional features.
3. LDA is a parametric method (it assumes unimodal Gaussian likelihoods).
4. If the distributions are significantly non-Gaussian, the LDA projections may not preserve complex structure in the data needed for classification.
5. LDA will also fail if discriminatory information is not in the mean but in the variance of the data.

A popular variant of LDA are the **Multi-Layer Perceptrons** (or MLPs).

Here is **FLDA / LDA explained in SUPER SIMPLE language**, like you're learning it for the first time. No math fear. No heavy formulas. Just pure intuition 👇 🔥

⭐ WHAT IS LDA (Fisher's Linear Discriminant Analysis)?

LDA is a **supervised dimensionality reduction** method.

Meaning:

👉 It reduces features **using class labels** 👉 Unlike PCA (which ignores labels)

LDA tries to find **the best direction** to project data so that **different classes are separated as much as possible**.

⭐ THE GOAL OF LDA (simple version)

LDA wants to find a line (or direction) where:

✔️ Class means are **far apart** (good separation)

✔️ Each class's points are **close to each other** (compact clusters)

→ This gives **maximum class separation**

So:

📌 **Between-class distance should be big** 📌 **Within-class distance should be small**

⭐ HOW LDA DOES IT (baby version)

Imagine you have TWO classes:

- Class 1 (blue points)
- Class 2 (red points)

LDA looks for a direction **w** (a vector) such that:

1️⃣ When you project points onto this line…

- Blue points gather together
- Red points gather together
- The groups are far from each other

2️⃣ So your classification becomes easier

→ You just draw a threshold between the two projected groups.

⭐ WHAT ARE THE MEANS?

Each class has a mean point:

- Mean of class 1 = average of blue points
- Mean of class 2 = average of red points

LDA uses the difference between these means to separate classes.

⭐ WHAT IS WITHIN-CLASS VARIANCE?

This describes how **spread out** each class is.

We want it:

❌ NOT too large ❌ NOT overlapping ✔️ as tight as possible

Tight clusters = easier classification.

⭐ THE KEY IDEA

LDA solves this optimization:

```
Make (distance between class means) big
And (spread inside each class) small
```

Mathematically:

```
J(w) = (between-class variance) / (within-class variance)
```

LDA **maximizes this ratio** to find the best direction **w**.

⭐ MAGIC RESULT (Final formula)

The perfect direction w* is:

```
w* = Sw^-1 (m2 - m1)
```

Meaning:

👉 You take difference of class means 👉 Multiply with inverse of within-class scatter 👉 You get the best projection direction

You don't need to compute it by hand—scikit-learn does it.

⭐ WHAT IS SB AND SW?

In simple words:

- **SB (between-class scatter)** = how far the class centers are
- **SW (within-class scatter)** = how spread each class is

LDA wants:

- **SB BIG**
- **SW SMALL**

⭐ LDA vs. PCA (in 5 seconds)

PCA	LDA
Unsupervised	Supervised
Maximizes data variance	Maximizes class separation
Finds directions of max spread	Finds directions separating classes
Ignores labels	Uses labels

⭐ LIMITATIONS (simple)

1. LDA can produce at most **C–1 features** for C classes (If 40 classes → max 39 LDA dimensions)
2. It assumes data in each class is Gaussian → If not, results may be worse.
3. If classes overlap a lot → LDA won't help much.
4. If class covariance is not equal → LDA is not optimal.

⭐ SUPER EASY SUMMARY (learn this!)

LDA finds the best direction to separate classes by:

- Making class means far apart
- Making classes internally tight
- Using class labels
- Producing up to C–1 features
- Giving great low-dimensional representations for classification

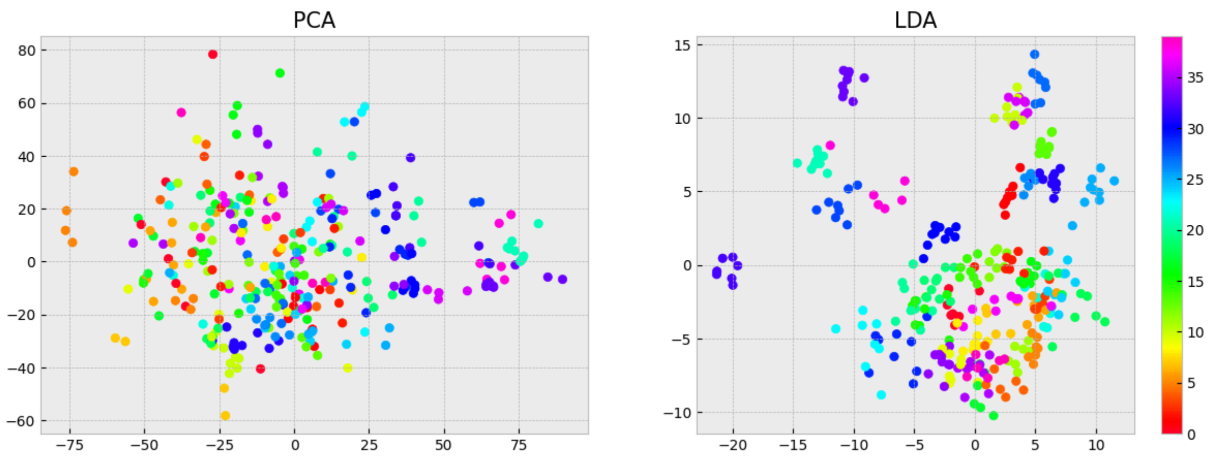
---

If you want, I can make a **1-slide summary**, **handwritten-style notes**, or a **visual example** with a picture!

In [44]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
```

In [45]:
```
# Dimensionality Reduction: PCA
pca = Pipeline([('scaler', StandardScaler()),
                ('pca', PCA(n_components=2))])
pca.fit(X_train)
y_train_pca = pca.transform(X_train)

# Dimensionality Reduction: LDA
lda = Pipeline([('scaler', StandardScaler()),
                ('lda', LDA(n_components=2))])
lda.fit(X_train, t_train)
y_train_lda = lda.transform(X_train)
```

In [46]:
```
plt.figure(figsize=(15,5))
plt.subplot(1,2,1); plt.title('PCA', fontsize=15)
plt.scatter(y_train_pca[:,0], y_train_pca[:,1], c=t_train, cmap=plt.cm.gist_rainbow
plt.subplot(1,2,2); plt.title('LDA', fontsize=15)
plt.scatter(y_train_lda[:,0], y_train_lda[:,1], c=t_train, cmap=plt.cm.gist_rainbow
plt.colorbar();
```



Manifold Learning (Non-Linear Dimensionality Reduction)

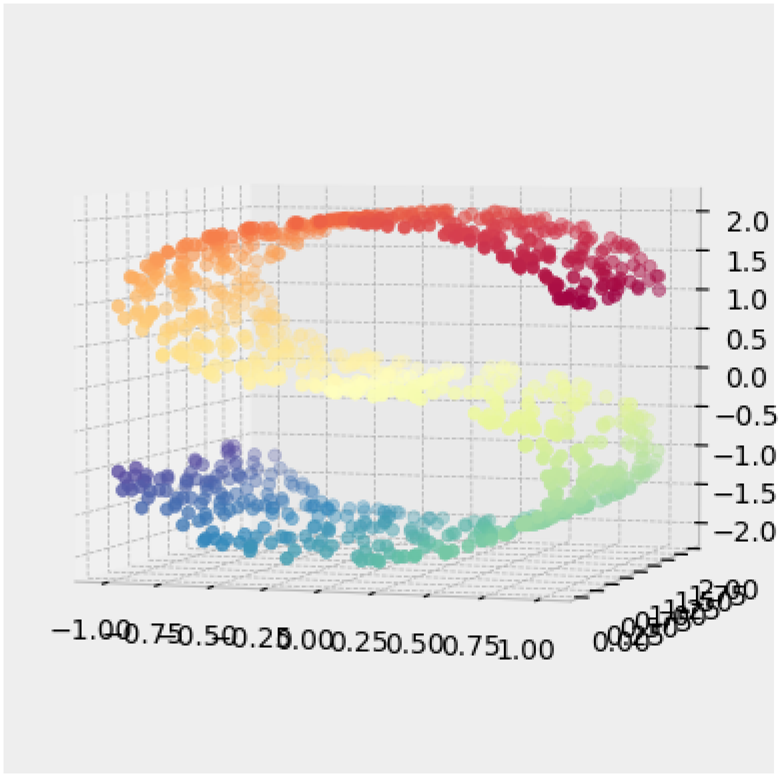
As we have already noted, many natural sources of data correspond to low-dimensional, possibly noisy, non-linear manifolds embedded within the higher dimensional observed data space.Capturing this property explicitly can lead to improved density modeling compared with more general methods.

PCA and LDA are often used to project a data set onto a lower-dimensional space. However both of them assume that the data samples live in an underlying linear manifold.

There are other dimensionality reduction techniques that do not assume the manifold is linear. They include:

1. Multi-Dimensional Scaling (MDS)
2. Isometric Mapping (ISOMAP)
3. Locally Linear Embedding (LLE)
4. t-Distributed Stochastic Neighbor Embedding (t-SNE)

In [47]:
```
from sklearn.datasets import make_s_curve

n_points = 1000
X_scurve, color = make_s_curve(n_points, random_state=0)
n_neighbors = 10
n_components = 2

fig=plt.figure(figsize=(15, 5))
ax=fig.add_subplot(111, projection='3d')
ax.scatter(X_scurve[:, 0], X_scurve[:, 1], X_scurve[:, 2], c=color, cmap=plt.cm.Spe
ax.view_init(4, -72)
```



Multi-Dimensional Scaling (MDS)

Another linear technique with a similar aim is **multidimensional scaling**, or **MDS**. It finds a low-dimensional projection of the data such as to **preserve the pairwise distances between data points**, and involves finding the eigenvectors of the distance matrix.

Consider a set of mean-centered observations $X = \{x_1, x_2, \ldots, x_N\}$ where $x_i \in \mathrm{R}^D$. By mean-centered samples X, I mean that $\mu_j = \sum_{i=1}^{N} x_{ij} = 0, \forall j = 1, 2, \ldots, D$.

Consider the **proximity matrix** D that stores pairwise distances of data points $d_{ij} = \text{distance}(x_i, x_j)$:

$$D = \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1N} \\ d_{21} & d_{22} & \cdots & d_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ d_{N1} & d_{N2} & \cdots & d_{NN} \end{bmatrix}$$

Note that D is an $N \times N$ symmetric matrix.

Given an assumed Euclidean proximity matrix, D, the **goal** of MDS is to find a set of points, Y, that have the same proximity matrix in an M-dimensional space, where $M < D$.

- MDS preserves the global data structure.
- MDS can use any distance metric to compute the pairwise distances between points.
- There is the need to store the proximity matrix (half of it, since it is symmetric). Thus requiring a significant computational and storage resources for large datasets.
    - There are $N(N-1)/2$ distance computations, where N is the number of samples.
- **Classical MDS** refers to MDS when using Euclidean distances.
- In the case where the pairwise distances are computed with the Euclidean distance metric, MDS gives equivalent results to PCA. Therefore, MDS is a generalization of PCA.