Unsupervised Learning

- No targets
- Why use it ?
    - Understand your features
    - Better use of features in supervised models

Plan

- Principal Components
    - Highly popular model for dimensionality reduction
- Clustering
    - K-means to cluster samples
    - Hierarchical clustering
- Recommender systems
    - Netflix prize
    - Pseudo SVD

Alternate basis

We can find an *alternate* set of n basis vectors of length n

$$\tilde{\mathbf{v}}_{(1)}, \ldots, \tilde{\mathbf{v}}_{(n)}$$

and translate $\mathbf{x^{(i)}}$ into coordinates $\tilde{\mathbf{x}}^{(\mathbf{i})}$ in the alternate basis

$$\mathbf{x^{(i)}} = \sum_{j=1}^{n} \tilde{\mathbf{x}}^{(\mathbf{i})}_j * \tilde{\mathbf{v}}_{(j)}$$

PCA finds alternate basis $\tilde{\mathbf{v}}_{(1)}, \ldots, \tilde{\mathbf{v}}_{(n)}$

- $\tilde{\mathbf{v}}_{(j)}$ is called *Principal Component* j
- That are mutually orthogonal
$$\tilde{\mathbf{v}}_{(j)} \cdot \tilde{\mathbf{v}}_{(j')} = 0, \text{for } j \neq j'$$
- $\tilde{\mathbf{v}}_{(j)}$ has more variation than $\tilde{\mathbf{v}}_{(j')}$ for $j' > j$,

The number of basis vectors in the original and alternate bases is both n.

Suppose we reduced the number of alternate basis vectors to $r \leq n$.

- We set $\tilde{\mathbf{x}}_j^{(\mathbf{i})} = 0$ for $j > r$

This is the *reduced dimension* approximation of $\mathbf{x}^{(\mathbf{i})}$.

$$\mathbf{x}^{(\mathbf{i})} = \sum_{j=1}^{r} \tilde{\mathbf{x}}_j^{(\mathbf{i})} * \tilde{\mathbf{v}}_{(j)}$$

Since the basis vectors are ordered such that $\tilde{\mathbf{v}}_{(j)}$ captures more variation than $\tilde{\mathbf{v}}_{(j')}$ for $j' > j$

- Dropping the alternate bases of higher index loss minimal information

So PCA is the process of

- Finding alternate bases $\tilde{\mathbf{v}}$
- The alternate bases capture correlation among original features $\mathbf{x}$
- Projecting $\mathbf{x^{(i)}}$ onto $\tilde{\mathbf{v}}$ to obtain transformed vector $\tilde{\mathbf{x}}^{(\mathbf{i})}$ of synthetic features
- Choosing an r so that $\tilde{\mathbf{x}}^{(\mathbf{i})}$ is of dimension $r \leq n$

Return to parent notebook

What is PCA

- a way to reduce dimensionality of features
    - Do we really need all 784 pixels in MNIST ?
    - Hedging in Fixed Income
- a way to cluster samples based on similarity of *features*
- contrast this to Decision Tree
    - also clusters samples with similar features
    - but guided by the clusters having same targets

PCA: High Level

TL;DR

- PCA is a technique for creating "synthetic features" from the original set of features
- The synthetic features may better reveal relationships among original features
- May be able to use reduced set of synthetic features (dimensionality reduction)
- Synthetic features as a means of clustering samples
- **All features need (and will be assumed to be) centered: zero mean**
- PCA is very scale sensitive; often normalize each feature to put on same scale

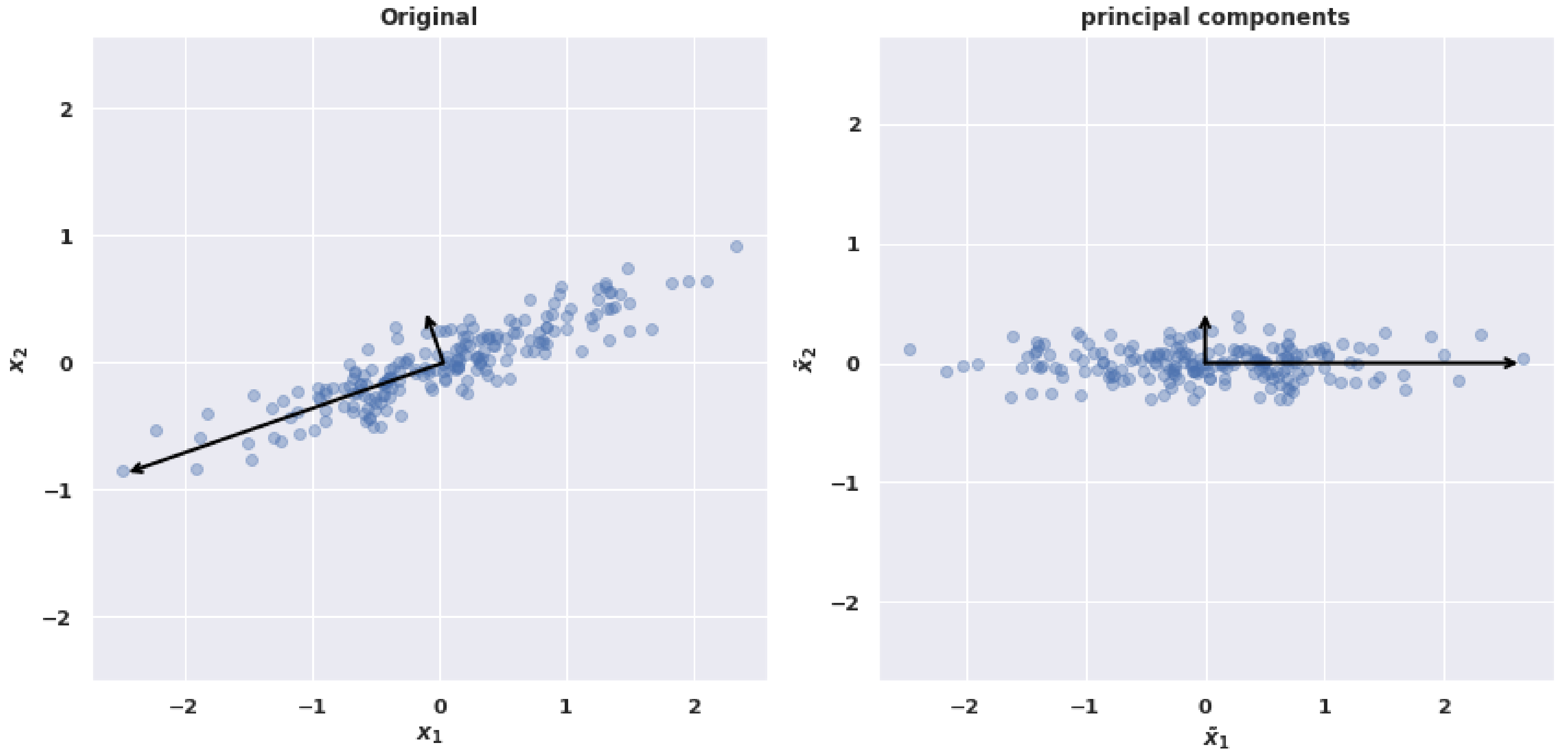
The key ideas behind PCA:

- Dimension reduction: from n "original" features to $n' \le n$ "synthetic" features
    - synthetic features are more like "concepts" than attributes
    - commonality of purpose
- Feature transformation: from correlated original features to *independent* synthetic features
- Order of "importance" of synthetic features
    - Drop the less important synthetic features: dimension reduction
    - Important relative to feature variation *not* prediction
        - unlike Decision Trees; there is no target

Preview

In one picture:

In [4]:
```
X = vp.create_data()

vp.show_2D(X)
```



The points in the left and right plots are the same, except for the coordinte system.

- Left plot: coordinate system is the horizontal and vertical axes, as usual
- Right plot: coordinate system is the dark, arrowed lines -- identical to the lines on the left plot

Bottom line: same date, expressed in a different way

- Left plot: features $\mathbf{x}_1, \mathbf{x}_2$
- Right plot: features $\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2$

In the left plot, we can clearly see that the data set's features $\mathbf{x}_1, \mathbf{x}_2$ are correlated.

In the right plot: $\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2$ are

- independent
- with $\tilde{\mathbf{x}}_1$ expressed greater variation

The arrowed vectors in the original

- correspond to the components: the alternate basis vectors
- the first component is in the negative direction
- so the plot on the right, which has this component in the *positive* direction, is "flipped"
    - we illustrate this by showing the mapping of the points in red below

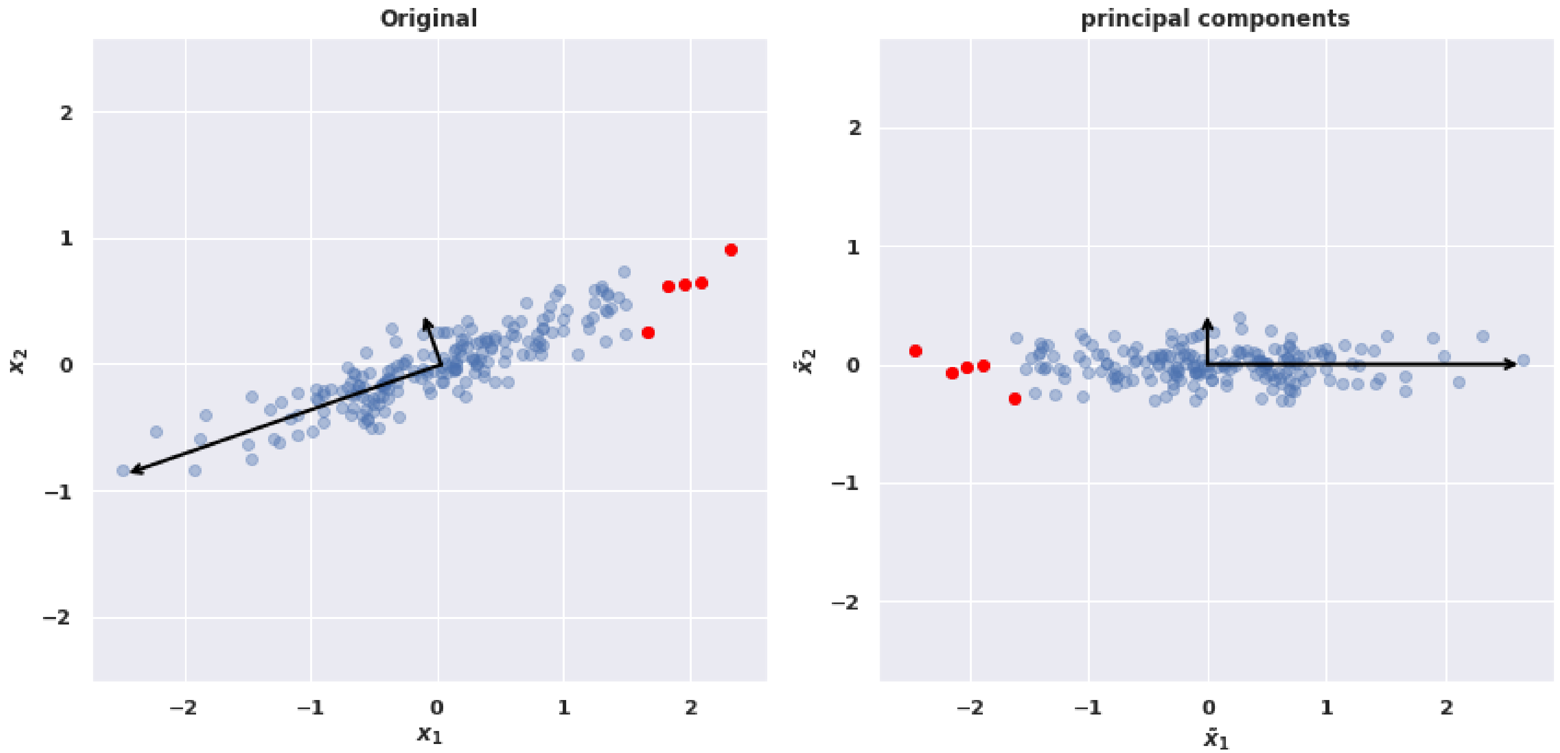
You can see that the alternate basis

- First basis vector (component) points left
- So the first coordinate (feature) in the alternate basis is opposite the first coordinate in the original

We can more easily see it by observing how the red examples behave

In [5]:

```
vp.show_2D(X, points=X[ X[:,0] > 1.5 ])
```



To summarize

$$\mathbf{X} = \tilde{\mathbf{X}} V^T$$

That is

- Examples $\mathbf{X}$
    - original features $\mathbf{x}^{(\mathbf{i})}$
- Re-expressed in new basis V^T
    - synthetic features $\tilde{\mathbf{x}}^{(\mathbf{i})}$
    - new basis $\backslash\mathrm{V}^T$ may have fewer dimensions $r \leq n$ than original

Return to parent notebook

PCA via Matrix factorization

$\mathbf{X} = \{\mathbf{x^{(i)}} \mid i = 1, \ldots, m\}$

No targets !

$\mathbf{X}$ is **zero centered** (subtract mean from each feature)

Our goal is to

- Find $\tilde{\mathbf{X}}, V^T$
- Such that
  $\mathbf{X} = \tilde{\mathbf{X}} V^T$

Decomposing $\mathbf{X}$ into a product (as above) is called *matrix factorization*

Some types of matrix factorization we'll mention

- Singular Value Decomposition
- Eigen Decomposition
- CUR Decomposition

We will first derive PCA using Singular Value Decomposition of $\mathbf{X}$

Singular Value Decomposition (SVD) Factorization

Factor matrix X into product of 3 matrices:

$$\mathbf{X} = U \Sigma V^T$$

- U: $m \times n$, columns are orthogonal unit vectors
  - $UU^T = I$
- Σ : $n \times n$ diagonal matrix

  $$\mathrm{diag}(\Sigma) = [\sigma_1, \sigma_2, \ldots, \sigma_n]$$
- V : $n \times n$, columns are orthogonal unit vectors
  - $VV^T = I$

Moreover, the diagonal elements of Σ are in descending order of magnitude

$$\sigma_j > \sigma_{j'} \quad \text{for } j' > j$$

Let V^T be the new basis vectors

- Since $VV^T = I$ these vectors are orthogonal

We need to find the synthetic features $\mathbf{X}$ relative to these bases

$$\mathbf{X} = \tilde{\mathbf{X}} V^T$$

$$
\begin{aligned}
\mathbf{X} &= \tilde{\mathbf{X}} V^T && \\
U\Sigma V^T &= \tilde{\mathbf{X}} V^T && \text{factorization } \mathbf{X} = U\Sigma V^T \\
U\Sigma V^T V &= \tilde{\mathbf{X}} V^T V && \text{multiple both sides by } V \\
U\Sigma &= \tilde{\mathbf{X}} && \text{since } V^T V = I
\end{aligned}
$$

Thus SVD gives us both $\tilde{\mathbf{X}}$ and V^T as desired.

In fact since

$$\tilde{\mathbf{X}} = U\Sigma$$

there is interesting structure in $\tilde{\mathbf{X}}$

Recall that U is orthonormal

$$UU^T = I$$

so that its vectors are unit length

Since Σ is diagonal (all zero for non-diagonal elements)

- The diagonal of Σ, denoted
  $$\text{diag}(\Sigma) = [\sigma_1, \sigma_2, \ldots, \sigma_n]$$
  scales the colums of U
  $$\begin{aligned}
  \tilde{\mathbf{X}}^{(\mathbf{i})} &= (U\Sigma)^{(\mathbf{i})} \\
  &= U^{(\mathbf{i})} * \text{diag}(\Sigma) \\
  &= [U_1^{(\mathbf{i})} * \sigma_1, U_2^{(\mathbf{i})} * \sigma_2, \ldots, U_n^{(\mathbf{i})} * \sigma_n]
  \end{aligned}$$

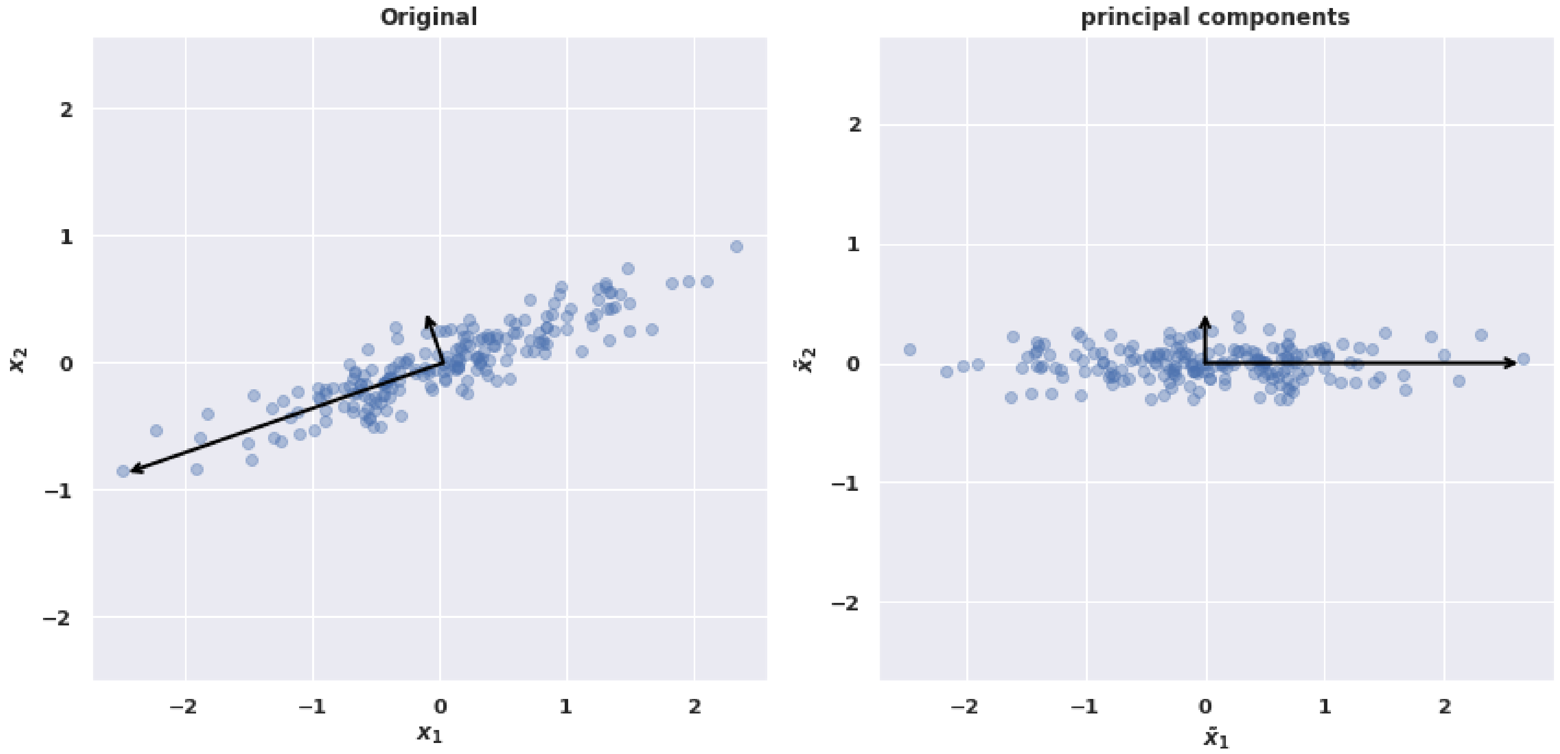
Thus

- U is a "standardized" version of features $\tilde{\mathbf{X}}$
    - unit standard deviation
- $\tilde{\mathbf{X}} = U\Sigma$ is the non-standardized features

A picture may clarify the distinction between the standardized and non-standardized $\tilde{\mathbf{X}}$.

Here is the non-standardized $\tilde{\mathbf{X}}$ that we've seen previoulsy

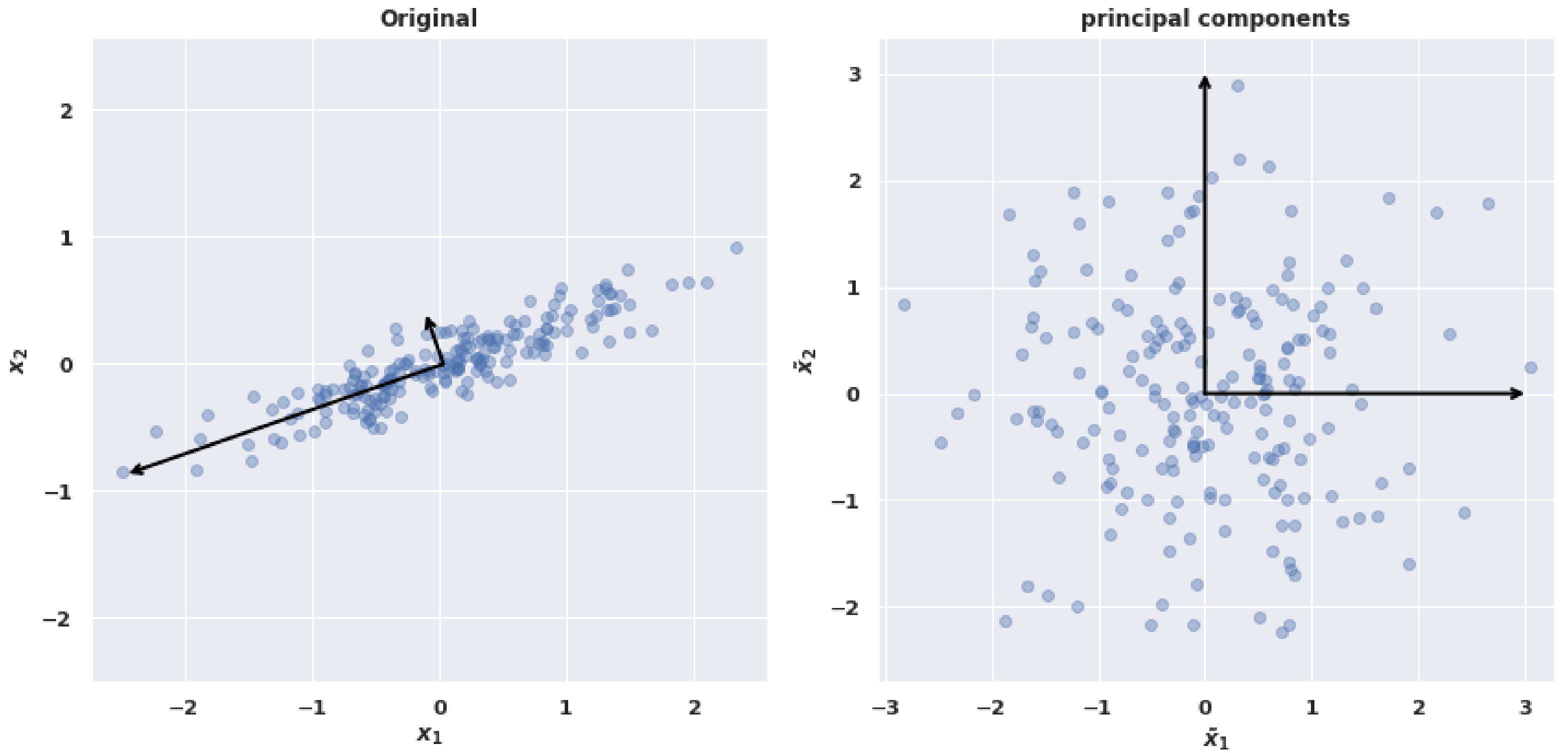
In [6]:
```
X = vp.create_data()

vp.show_2D(X)
```



And here is the standardized plot

- The length of each basis vector is 1
- Rather than σ_j
- By stretching each component by σ_j we recover the non-standardized plot

In [7]:
```
vp.show_2D(X, whiten=True)
```



So

- The synthetic features $\tilde{\mathbf{X}}$
- Are "standardized" synthetic features U
- Scaled by $\text{diag}(\Sigma)$

Return to parent notebook

Dimensionality reduction

Thus far we have exactly replicated $\mathbf{X}$ via new bases V^T

$$\mathbf{X} = \tilde{\mathbf{X}} V^T$$

$\tilde{\mathbf{X}}$ is the same dimensions as $\mathbf{X}$, so each example is of length n in both the original and alternate representation.

We will now change $\tilde{\mathbf{X}}$ to $(m \times r)$ for $r \leq n$.

That is: the alternate representation may be of reduced dimension.

Recall that

$$\begin{aligned}\tilde{\mathbf{X}}^{(\mathbf{i})} &= (U\Sigma)^{(\mathbf{i})} \\ &= U^{(\mathbf{i})} * \mathrm{diag}(\Sigma) \\ &= [U_1^{(\mathbf{i})} * \sigma_1, U_2^{(\mathbf{i})} * \sigma_2, \ldots, U_n^{(\mathbf{i})} * \sigma_n]\end{aligned}$$

By setting

$$\sigma_j = 0, \text{ for all } j > r$$

we zero out all features with index exceeding r

$$\begin{aligned}\tilde{\mathbf{X}}'^{(\mathbf{i})} &= (U\Sigma)^{(\mathbf{i})} \\ &= [U_1^{(\mathbf{i})} * \sigma_1, U_2^{(\mathbf{i})} * \sigma_2, \ldots, U_r^{(\mathbf{i})} * \sigma_r, \mathbf{0}, \ldots \mathbf{0}]\end{aligned}$$

The dimensions of $\tilde{\mathbf{X}}'^{(\mathbf{i})}$ is effectively reduced from n to $r \leq n$.

Zeroing out the diagonal elements of Σ with index $j > r$ makes the values in

- The columns of U with index $j > r$
- The rows of (V^T) with index $j > r$ irrelevant.

We can therefore write

$$\mathbf{X}' \approx \mathbf{X}$$

where

$$\mathbf{X}' = U'\Sigma'(V^T)'$$

where

$\mathbf{X}$ and $\mathbf{X}'$ have the *same* dimensions, but the values in $\mathbf{X}'$ can only *approximate* the values in $\mathbf{X}$.

The advantage is that $\tilde{\mathbf{X}}'$ is of lower dimension.

Best lower rank approximation of $\mathbf{X}$

We could have reduced the dimension of $\tilde{\mathbf{X}}'$ by dropping *any* set of $(n - r)$ columns.

Let D denote the set of size $(n - r)$ containing the indexes of the columns we choose to drop.

Is there a particular reason for dropping the columns

$$D = \{j | j > r\}$$

To answer the question, we first define the *error* of the approximation $\mathbf{X}'$ relative to the true $\mathbf{X}$

$$||\mathbf{X}' - \mathbf{X}||_2 = \sum_{i,j} (\mathbf{X}'_{i,j} - \mathbf{X}_{i,j})^2$$

The above is called the Froebenius Norm (and looks like MSE in form).

The "best" set of columns to drop (from Σ) are the one resulting in the *lowest* error.

Recall that

$$\tilde{\mathbf{X}}^{(\mathbf{i})} \quad = \quad [U_1^{(\mathbf{i})} * \sigma_1, U_2^{(\mathbf{i})} * \sigma_2, \ldots, U_n^{(\mathbf{i})} * \sigma_n]$$

and

$$\mathbf{X}^{(\mathbf{i})} = \tilde{\mathbf{X}}^{(\mathbf{i})} V^T$$

So

$$\begin{aligned}
\mathbf{X}_j^{(\mathbf{i})} &= [U_1^{(\mathbf{i})} * \sigma_1, U_2^{(\mathbf{i})} * \sigma_2, \ldots, U_n^{(\mathbf{i})} * \sigma_n] \cdot (V^T)_j \qquad \text{where } (V^T)_j \text{ is column} \\
&= \textstyle\sum_{k=1}^n U_k^{(\mathbf{i})} * \sigma_k * (V^T)_j^{(k)} \\
&= \textstyle\sum_{k=1}^n \sigma_k * (U_k^{(\mathbf{i})} * (V^T)_j^{(k)})
\end{aligned}$$

The approximation error in $\mathbf{X}_j^{(\mathbf{i})}$ induced by dropping the columns in D is

$$(\mathbf{X}_j^{(\mathbf{i})} - \mathbf{X}'^{(\mathbf{i})})^2 \quad = \quad \left(\sum_{k \in D} \sigma_k * (U_k^{(\mathbf{i})} * (V^T)_j^{(k)})\right)^2$$

Because the diagonal elements of Σ are in decreasing order of magnitude

$$\sigma_j > \sigma_{j'} \quad \text{for } j' > j$$

choosing D to be

$$D = \{j | j > r\}$$

results in dropping terms $U_k^{(\mathbf{i})} * (V^T)_j^{(k)}$ that are scaled by the $(n - r)$ *smallest* values of σ_k.

Although this is not mathematically precise, hopefully this provides some intuition as to why choosing D this way is a good idea.

Aside

It will turn out that

$$\begin{aligned} & \left(\sum_{k\in D} \sigma_k * (U_k^{(\mathbf{i})} * (V^T)_j^{(k)})\right)^2 \\ = \quad & \sum_{k\in D} \sigma_k^2 \end{aligned}$$

which makes the intuitive argument precise.

How many dimensions to keep ?

Since the diagonal elements of Σ are ordered

- We keep a cumulative sum of σ^2, which will sum to 1.
- Choose to keep the first k synthetic features
    - where the cumulative sum up to (and including) k is greater than some fraction
        - e.g., 95%.

We will illustrate this in the following example.

Return to parent notebook

The inverse transformation

We have shown how to transform original features $\mathbf{X}$ to synthetic features $\tilde{\mathbf{X}}$.

How about inverting the tranformation: recover $\mathbf{X}$ from $\tilde{\mathbf{X}}$?

Since

$$\begin{array}{lll}
\mathbf{X} & = & \tilde{\mathbf{X}}V^T & \text{definition} \\
\mathbf{X}V & = & \tilde{\mathbf{X}}V^TV & \text{multiply both sides by } V \\
\mathbf{X}V & = & \tilde{\mathbf{X}} & \text{since } V^TV = I
\end{array}$$

So

- V transforms from original features $\mathbf{X}$ to synthetic feature $\tilde{\mathbf{X}}$
- V^T transforms synthetic features $\tilde{\mathbf{X}}$ to original features $\mathbf{X}$

Return to parent notebook

Example: Reconstructing $\mathbf{x}$ from $\tilde{\mathbf{x}}$ and the principal components

It may be helpful to visualize

- The transformation from example features $\mathbf{x^{(i)}}$
- To synthetic features $\tilde{\mathbf{x}}^{\mathbf{(i)}}$

We will use a subset ofthe "smaller digits" (8×8) data

In [8]:

```
subset1 = [ 0, 4, 7, 9 ]
rh_digits = unsupervised_helper.Reconstruct_Helper( subset=[])
rh_digits.create_data_digits(subset=subset1)
```

In [9]:

```
n_components = 8
_= rh_digits.fit(n_components=n_components)
```

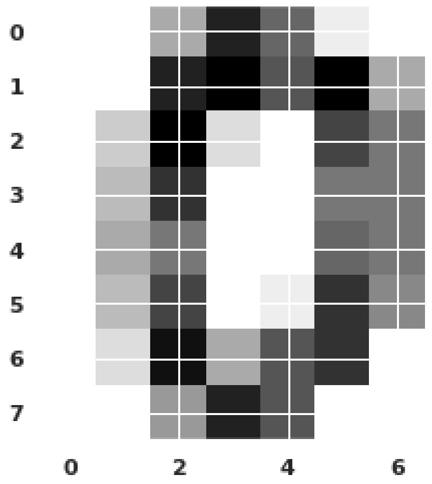
In [10]:
```
# Which example to show
data_idx = 0
fig0, ax0, figm, axm, figc, axc = rh_digits.show_data_comp(data_idx=data_idx)

plt.close(fig0)
plt.close(figm)
plt.close(figc)
```

Here is one example:

In [11]:

```
fig0
```

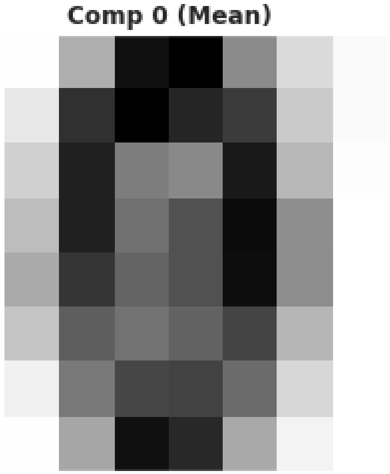
Out[11]:



And here is the "mean" of $\mathbf{X}$

- Recall: we assume $\mathbf{X}$ has been zero-centered
- So the mean is subtracted before perfroming PCA
- Which means it has to be *added* to the reconstructed image

In [12]:
```
figm
```
Out[12]:

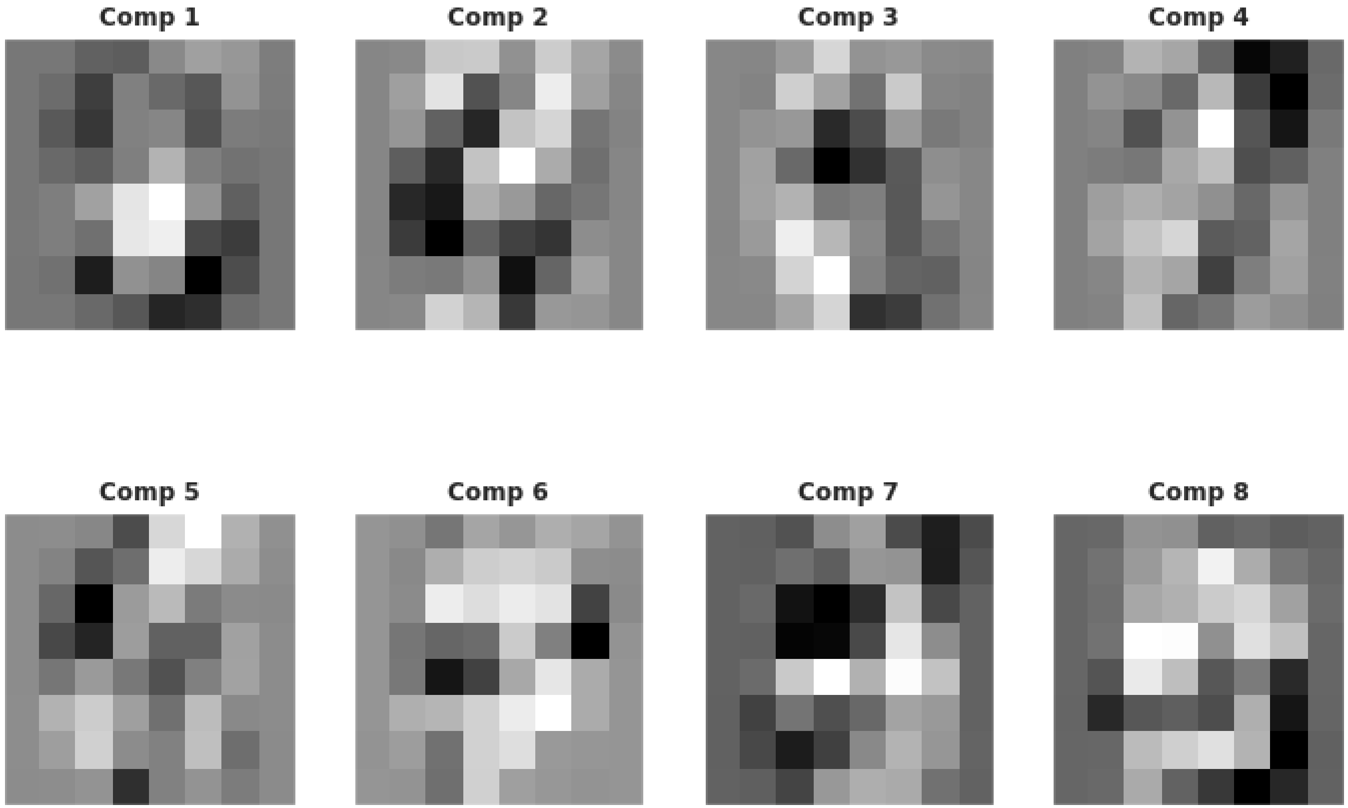


And here are the Principal Components (new bases)

- We performed PCA with a reduced number of components ($r < n$)
- There are `n_components` such basis vector
- *Each* component is of length n, but there are only $r < n$ components

In [13]:
```
figc
```

Out[13]:



It's not necessarily easy to interpret the components, particularly when n is large

- Component 1 *might* be the "concept" corresponding to the digit 0
- Component 4 *might* be the "concept" corresponding to the digit 4
- The other components *might* be *partial* shape concept, rather than entire digits

Let's progressively examine

$$\mathbf{x}^{(\mathbf{i})} = \sum_{j=1}^{r} \tilde{\mathbf{x}}_j^{(\mathbf{i})} * (V^T)^{(j)}$$

It may be helpful to remind ourselves of the shapes of each element in the equation

- $\mathbf{x^{(i)}}$ is of length n
- The components V^T are $(r \times n)$
    - Each component is of length n
    - There are $r < n$ components (e.g., $r =$ `n_components`)
- $\tilde{\mathbf{x}}^{(\mathbf{i})}$ is of length r

Note

- We treat the "mean\X$ as component 0
- With weight 1

So we construct an approximation of $\mathbf{x^{(i)}}$

- By adding weighted components $(V^T)^{(j)}$, each of length n
- The weight associated with component j is $\tilde{\mathbf{x}}_j^{\mathbf{(i)}}$
- So the sum is of length n

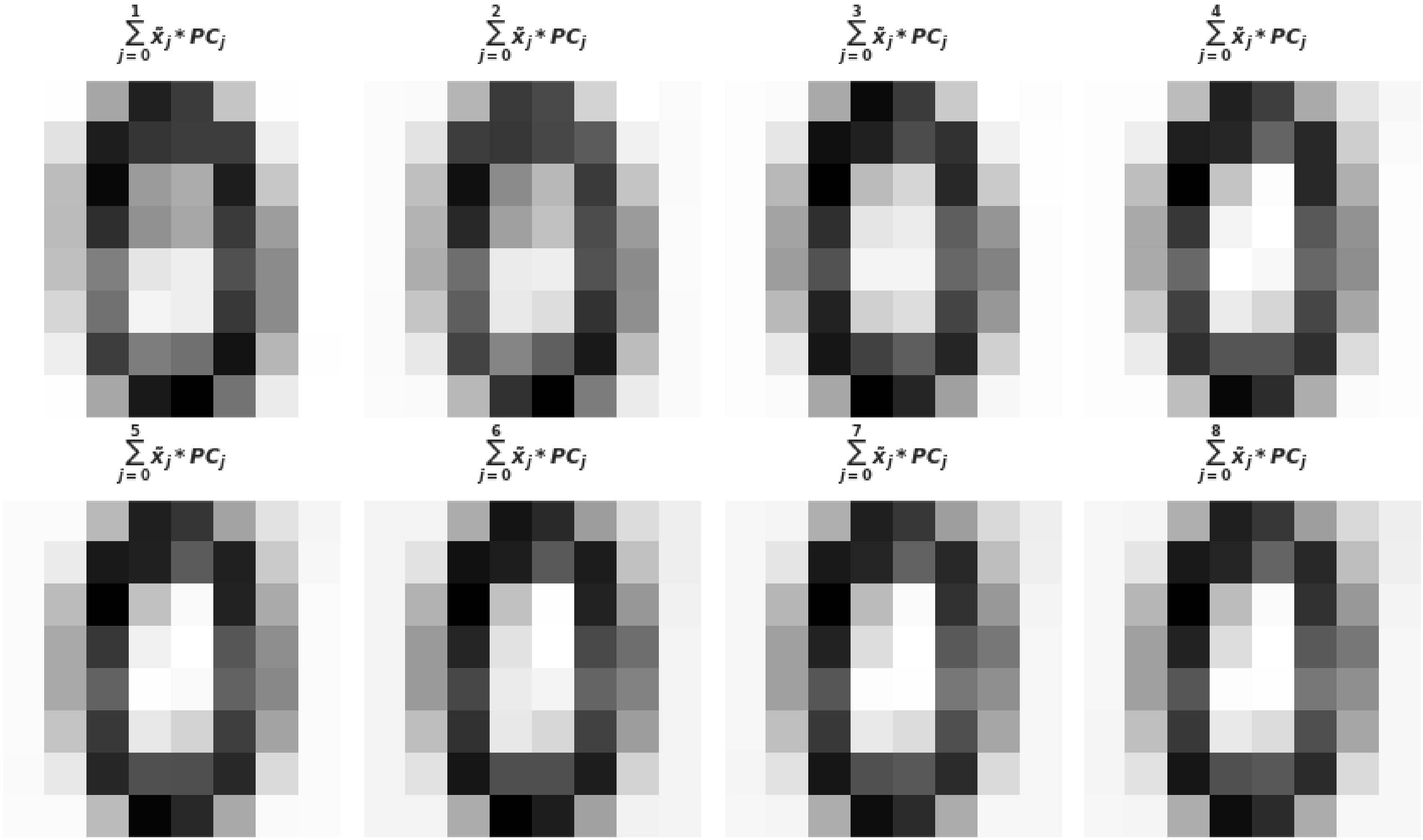
In [14]:

```
figi, axsi, fig_comp, axs_comp, x_tilde, error = rh_digits.show_recon(data_idx=d
ata_idx)
plt.close(figi)
plt.close(fig_comp)
```

In [15]:

```
figi
```

Out[15]:



You can see that the approximation using just the first component (and the mean)

- Is already a good approximation of $\mathbf{x}^{(\mathbf{i})}$
- Somewhat confirming our *guess* that component 1 represents the concept 0

We can confirm this by looking at $\tilde{\mathbf{x}}^{(\mathbf{i})}$

In [16]:

```python
print("x tilde = ", x_tilde)

arg_max = np.argmax(x_tilde)
print("Largest feature at index {idx:d}".format(idx=arg_max+1))
```

```
x tilde =  [ 18.94186713   5.09555081 -11.1175355    6.39727115  -0.96303788
   3.23324156   2.52928063   0.05886744]
Largest feature at index 1
```

As you can see, the magnitude of $\tilde{\mathbf{x}}_1^{(\mathbf{i})}$ is the largest among $\{\tilde{\mathbf{x}}_j^{(\mathbf{i})} | 1 \leq j \leq r\}$

In fact, we might try to confirm our intution

- By examining $\tilde{\mathbf{x}}^{(i')}$ for all i' where $\mathbf{y}^{(i')} = 0$ (assuming we have targets/lables)

In [17]:

```
# Get X tilde and the targets
Xtilde = rh_digits.dataProj
y = rh_digits.targets

# Filter to identify examples where target is equal to digit
digit = 0
mask = (y == digit)
Xtilde_digit = Xtilde[mask]
```

In [18]:
```python
print("x tilde, when y=0:")

for i in range(0,10):
    print( [ "{x:3.2f}".format(x=x_tilde_j) for x_tilde_j in Xtilde_digit[i] ])
```

```
x tilde, when y=0:
['18.94', '5.10', '-11.12', '6.40', '-0.96', '3.23', '2.53', '0.06']
['10.94', '11.59', '-8.82', '8.34', '-6.51', '4.21', '4.19', '-4.45']
['15.16', '8.46', '-9.70', '2.85', '-3.39', '-5.14', '-3.83', '-7.14']
['21.68', '9.93', '-12.65', '3.27', '1.31', '4.83', '-0.87', '-4.08']
['13.36', '8.83', '-11.29', '4.24', '-0.36', '-1.17', '0.93', '-12.11']
['19.57', '7.25', '-9.87', '-3.35', '-1.75', '7.13', '4.02', '0.30']
['20.21', '9.81', '-7.81', '-2.10', '-2.19', '-1.38', '-0.80', '3.83']
['10.75', '12.33', '-9.70', '-1.01', '-4.82', '-7.09', '-4.90', '-12.84']
['17.96', '12.42', '-5.09', '-5.08', '2.48', '-8.21', '-2.87', '-11.05']
['22.48', '1.51', '-7.25', '-10.33', '3.19', '3.57', '0.20', '4.89']
```

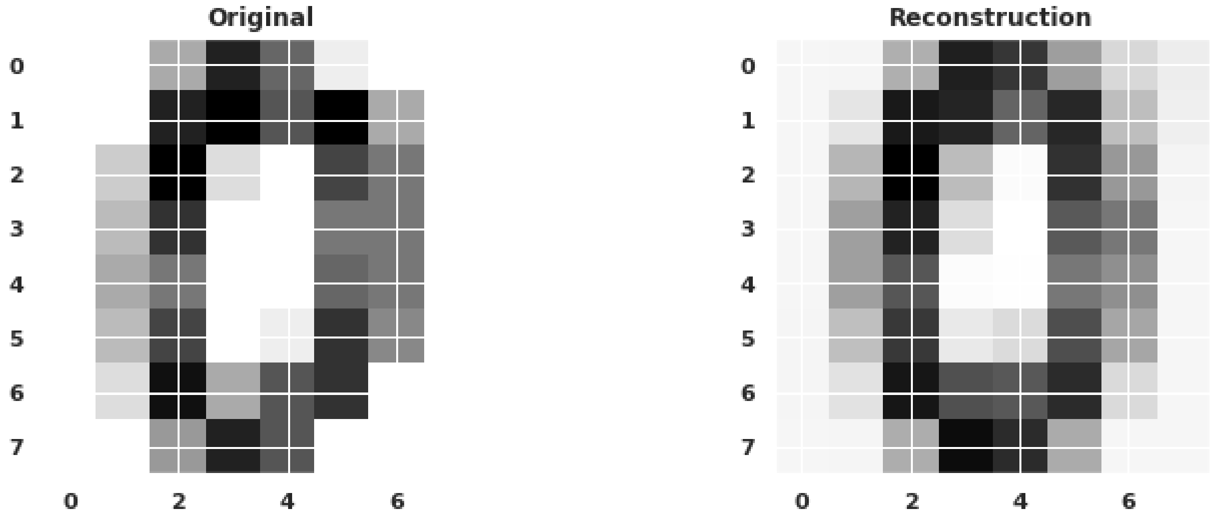
As you can see

- For examples $\mathbf{x}^{(i')}$ where $\mathbf{y}^{(i')} = 0$
- $\tilde{\mathbf{x}}^{(i')}$ is the largest value in $\tilde{\mathbf{x}}^{(i')}$

Here is a comparison of the original $\mathbf{x}^{(\mathbf{i})}$ and its reconstructed approximation

In [19]:
```
fig_comp
```

Out[19]:



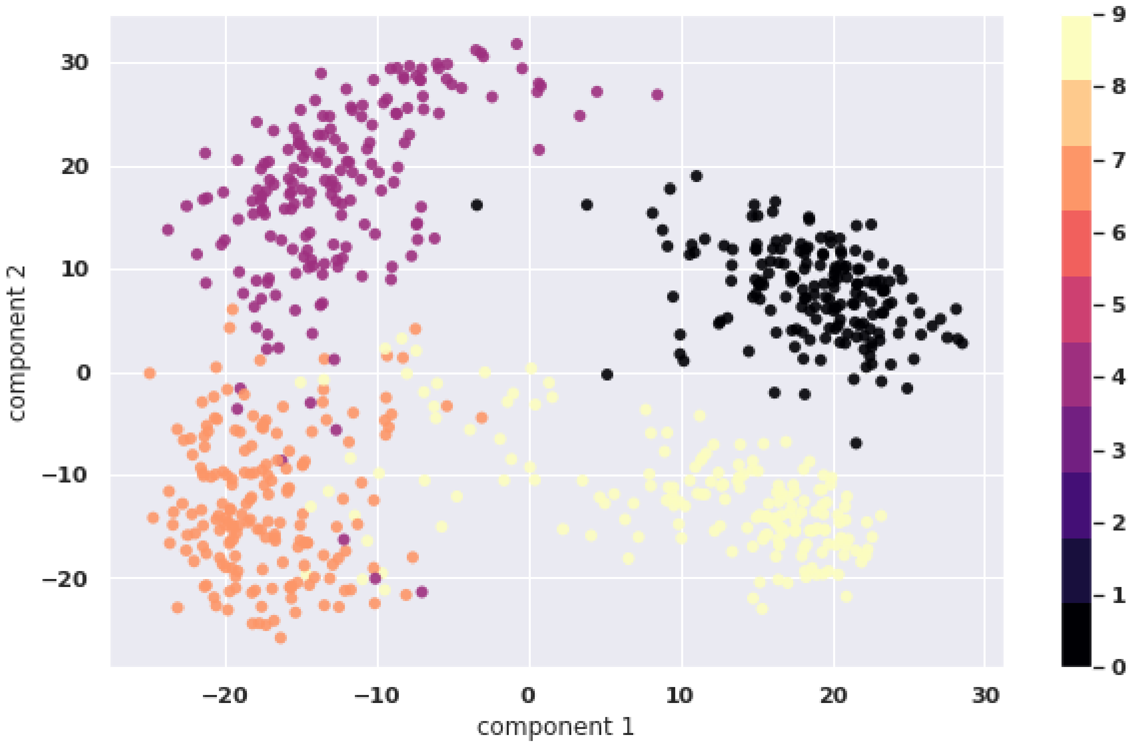
We could try to plot all our examples in r dimensionsal space

- To see whether examples formed clusters (examples with similar feature vectors of length r

But $r =$ `n_components` is too large in our case; let's just plot using the first two featues of $\tilde{\mathbf{x}}$

In [20]:
```python
vpt = unsupervised_helper.VanderPlas()

print("Number of examples: {n:d}".format(n=Xtilde.shape[0]))
vpt.digits_subset_show_clustering(Xtilde, y, save_file="/tmp/digits_subset_cluster.jpg" )
```

```
Number of examples: 718
```



Since we have targets/labels available (not generally the case for unsupervised learning)

- We can color the points according to their target
- We see that the 4 digits in the restricted examples cluster according to their features in $\tilde{\mathbf{x}}$
- Digit "0" is associated with (high $\tilde{\mathbf{x}}_1$, high $\tilde{\mathbf{x}}_2$)
- Digit "4" is associated with (low $\tilde{\mathbf{x}}_1$, high $\tilde{\mathbf{x}}_2$

Return to parent notebook

Dimensionality reduction:examples

MNIST example

- 784 features
    - are some redundant ? Can we capture "essence" with fewer pixels ?
    - Consider blocks of black pixels in 4 corners
        - pixel (i,j) highly correlated (across samples) with pixel (i+1,j), (i-1, j), (i, j+1), etc.
            - i.e. in many samples: when pixel(i,j) is black, so are surrounding pixels
    - If we replaced the block with 1 synthetic feature ("block of black in upper left ...")
        - we can reduce number of features (many pixels reduced to single)
        - reconstruction from compressed feature space to original preserves most info

So above goal was to reduce number of dimensions without losing info

Ideally, the reduction would be to a small enough (2 or 3) number of synthetic features

- that we could plot the samples in the tranformed synthetic feature space.

Retrieve the full MNIST dataset (70K samples)

We had previously used only a fraction in order to make our demo faster.

In [21]:

```
ush = unsupervised_helper.PCA_Helper()

X_mnist, y_mnist = ush.mnist_init()
```

```
Retrieving MNIST_784 from cache
```

In [22]:
```python
from sklearn.model_selection import train_test_split
X_mnist.shape, y_mnist.shape
X_mnist_train, X_mnist_test, y_mnist_train, y_mnist_test = train_test_split(X_mnist, y_mnist)
X_mnist_train.shape
```

Out[22]:
```
((70000, 784), (70000,))
```

Out[22]:
```
(52500, 784)
```

Perform PCA.

In [23]:

```
pca_mnist = ush.mnist_PCA(X_mnist_train)
```

In [24]:

```
pca_mnist.n_components_
X_mnist_train_reduced = ush.transform(X_mnist_train, pca_mnist)
X_mnist_train_reduced.shape
```

Out[24]:

```
153
```

Out[24]:

```
(52500, 153)
```

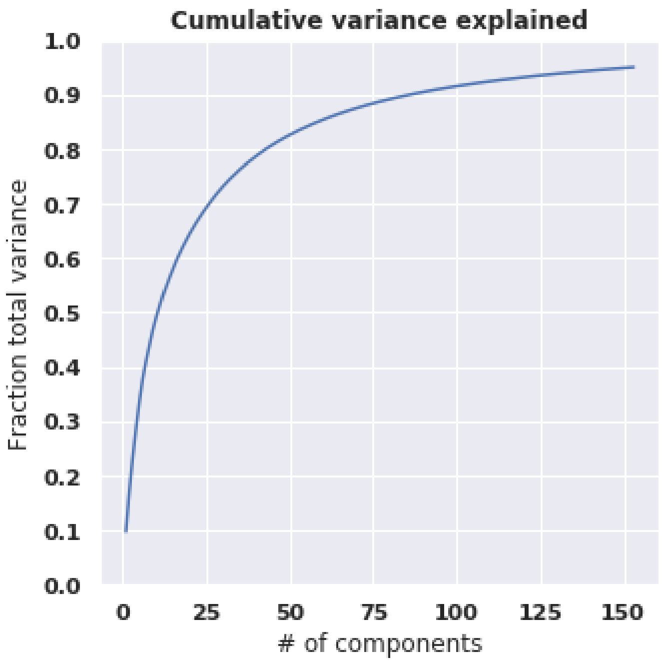
Let's plot the cumulative variance as a function of number of synthetic features.

This can help us determine how many synthetic features to keep.

In [25]:

```
_ = ush.plot_cum_variance(pca_mnist)

variance_goal_pct = 95
features_for_goal = ush.num_components_for_cum_variance(pca_mnist, .01 * varianc
e_goal_pct)
print("To capture {f:d}% of variance we need {d:d} synthetic features.".format(f
=variance_goal_pct, d=features_for_goal))
```

```
To capture 95% of variance we need 153 synthetic features.
```



So we need only about 20% of the original 784 features to capture 95% of the variance.

We can invert the PCA transformation to go from synthetic feature space back to original features.

That is, we can see what the digits look like when reconstructed from only 154 synthetic features.

First, let's look at the original:

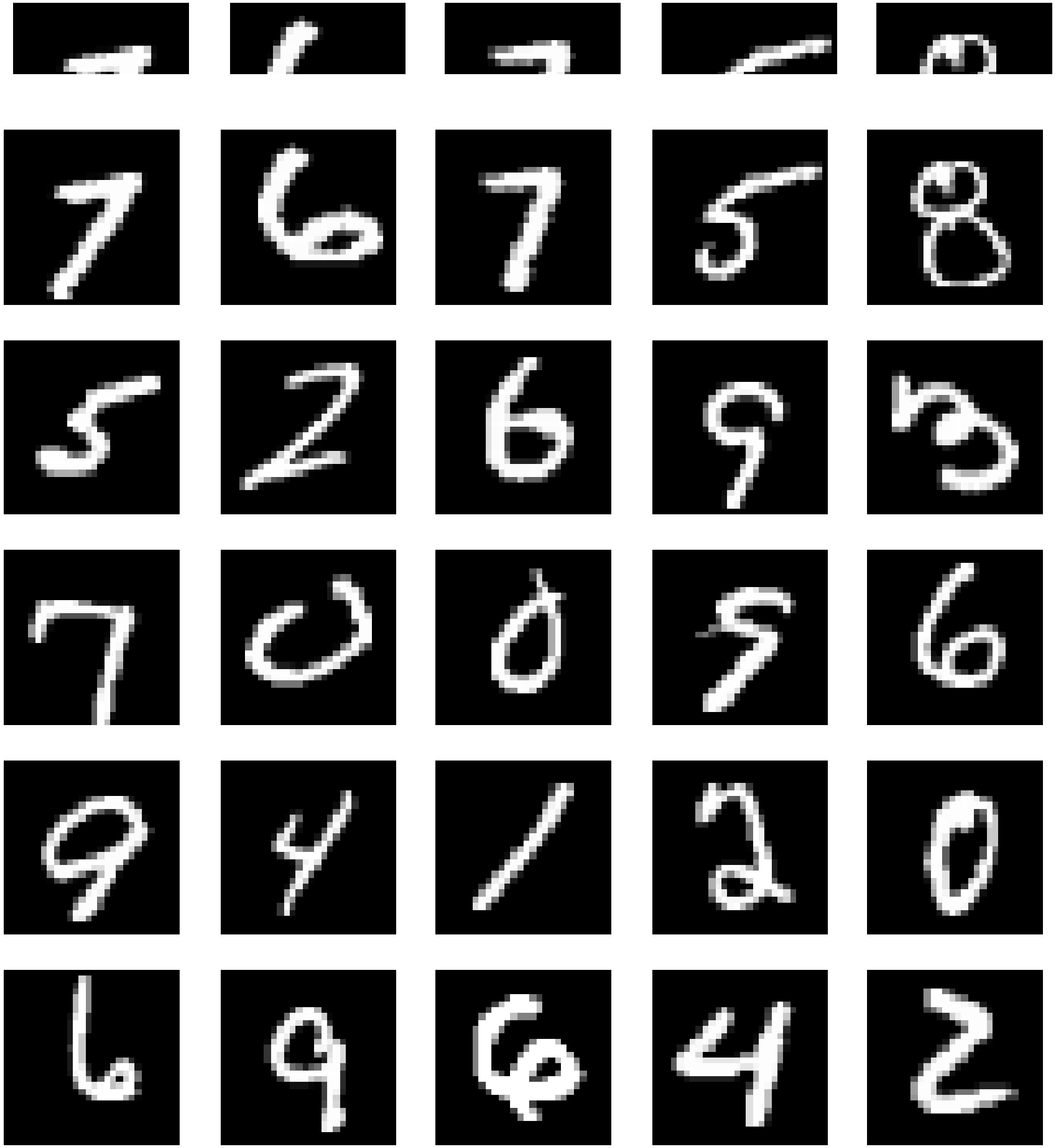
In [26]:

```
X_mnist_train_reduced = ush.transform(X_mnist_train, pca_mnist)
X_mnist_train_reduced.shape

# Show the original dataset
ush.mnh.visualize(X_mnist_train, y_mnist_train)
```

Out[26]:

```
(52500, 153)
```

Out[26]:

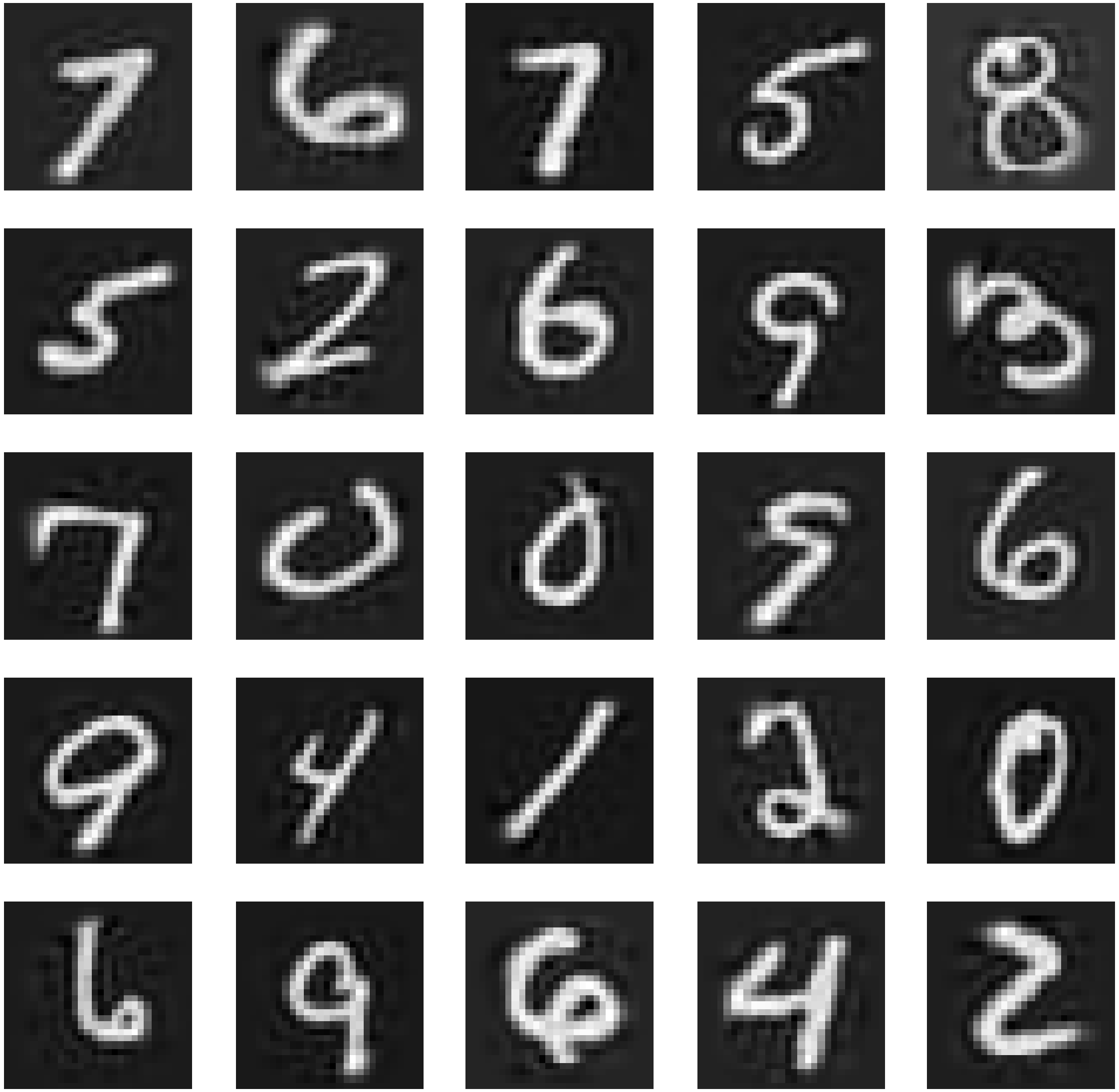


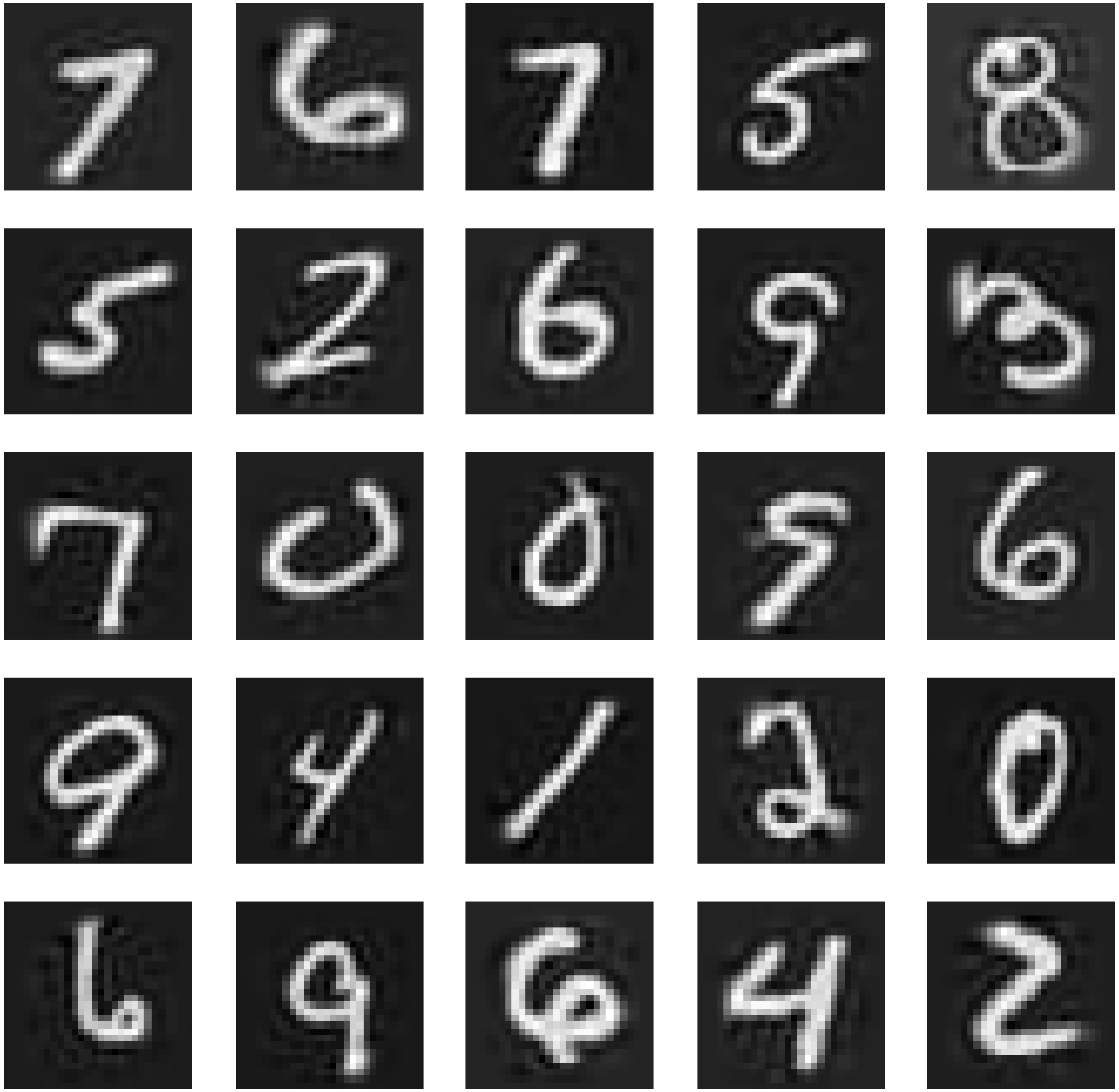
Next, the reconstructed

In [27]:

```
X_mnist_train_reconstruct = ush.inverse_transform(X_mnist_train_reduced, pca_mnist)
ush.mnh.visualize(X_mnist_train_reconstruct, y_mnist_train)
```

Out[27]:





A little fuzzy, but pretty good.

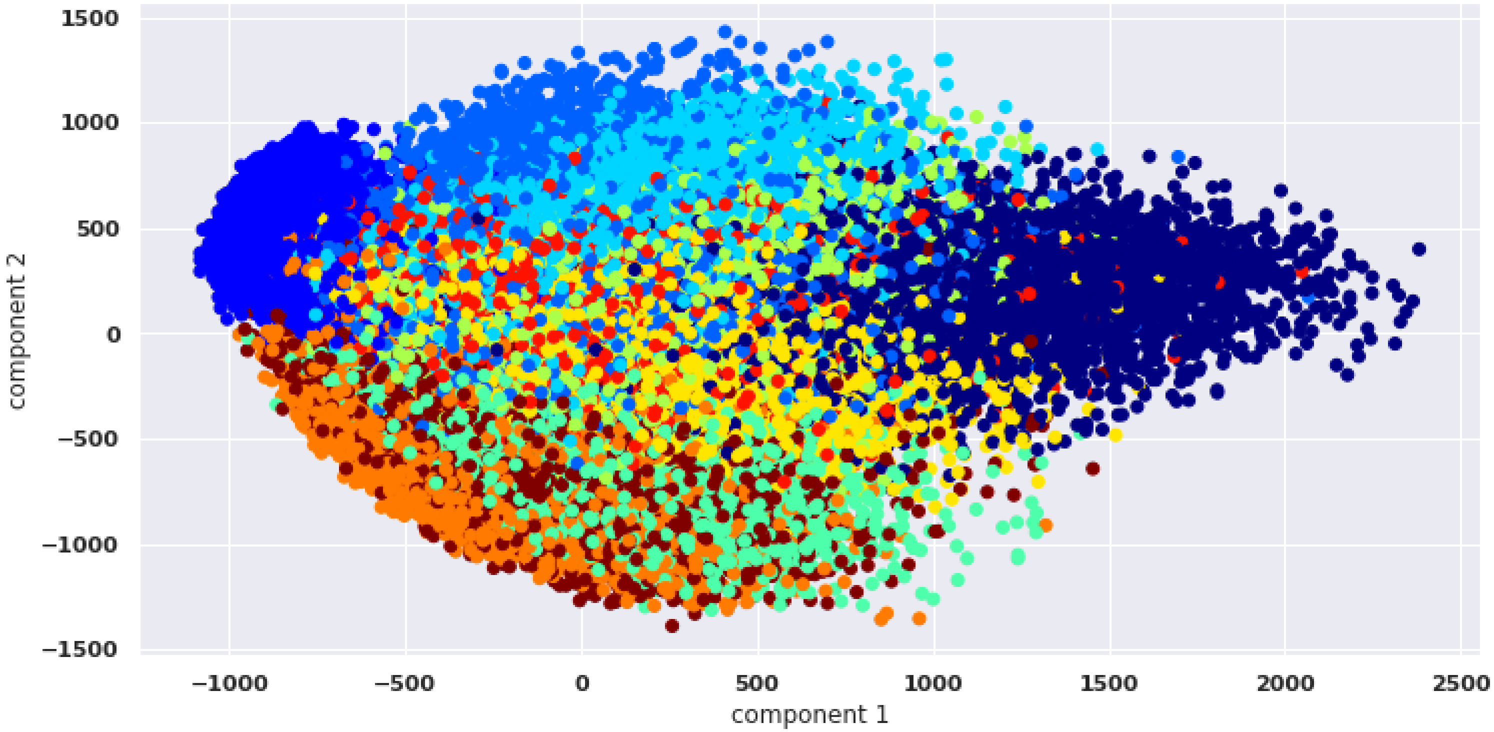
Suppose we retained only 2 synthetic features.

We would be able to plot each sample in two dimensions using the transformed coordinates.

Although targets are not necessary for PCA, here we do have labels associated with the images.

Let's plot the samples and color them according to their target.

In [28]:
```
_ = ush.mnist_plot_2D(X_mnist_train_reduced, y_mnist_train.astype(int))
```



Each color is a different digit.

You can see that the clustering is far from perfect

- but also suprisingly good considering we're using only 2 out of 784 features

Let's see how much variance is captured by only the first two synthetic features.

In [29]:

```
cumvar_mnist = np.cumsum(pca_mnist.explained_variance_ratio_)
first_comp =2
cumvar_first = cumvar_mnist[first_comp-1]

print("Cumulative variance of {d:d} PC's is {p:.2f}%, about {n:.1f} pixels".form
at(
    d=first_comp, p=100 *cumvar_first, n=cumvar_first * X_mnist_train.shape[1]))
```

```
Cumulative variance of 2 PC's is 16.87%, about 132.3 pixels
```

Is 17% good ? You bet !

With 784 original features (pixels)

- if each feature had equal importance, it would explain $1/784 = .12\%$ of the variance.

So the first synthetic feature captures the variance of 132 original features

- (assuming all were of equal importance).

Return to parent notebook

PCA in Finance

PCA of yield curve

[Litterman Scheinkman (https://www.math.nyu.edu/faculty/avellane/Litterman1991.pdf)](https://www.math.nyu.edu/faculty/avellane/Litterman1991.pdf)

This is one of the most important papers (my opinion) in quantitative Fixed Income.

It allows us to hedge a large portfolio of bonds with a handful of instruments.

Before we show the result: why is this an important advance in Finance ?

- Imagine we had a large portfolio of bonds with many maturites.
- A common goal in Fixed Income Finance is to *immunize* (hedge) a portfolio to changes in the Yield Curve.
- A simple way to construct the hedge is to
    - find the sensivitity of each bond in the portfolio to the $n = 14$ maturities
    - sum (over bonds in the portfolio) the individual bond sensitivities
    - minus 1 times resulting portfolio sensitivity is hedge that minimizes the portfolio's exposure to Yield Curve changes

But there are transaction costs (and complexity) with $n = 14$ bonds in the hedge portfolio.

Can we do nearly as well with $n' < n$ hedge bonds ?

That's exactly what PCA is designed for: dimensionality reduction.

- in this case, reducing the number of hedge bonds
- with minimal impact on immunization goal

Let's get the history of Yield Curves and look at first few samples.

In [30]:

```
ych = unsupervised_helper.YieldCurve_PCA()

# Get the yield curve data
data_yc = ych.create_data()
data_yc.head()
```

Out[30]:

	1M	2M	3M	6M	1J	2J	3J	4J	5J	6J	7J	8J	9J	10J
1992-02-29	0.0961	0.09610	0.0961	0.0958	0.0898	0.0864	0.0849	0.0837	0.0826	0.0817	0.0810	0.0806	0.0803	0.0804
1992-03-31	0.0970	0.09700	0.0970	0.0969	0.0912	0.0889	0.0877	0.0864	0.0852	0.0841	0.0833	0.0827	0.0823	0.0823
1992-04-30	0.0975	0.09750	0.0975	0.0975	0.0920	0.0892	0.0877	0.0862	0.0848	0.0837	0.0828	0.0822	0.0817	0.0816
1992-05-31	0.0978	0.09785	0.0979	0.0979	0.0920	0.0889	0.0874	0.0860	0.0847	0.0836	0.0828	0.0821	0.0817	0.0815
1992-06-30	0.0974	0.09745	0.0975	0.0975	0.0931	0.0904	0.0889	0.0874	0.0860	0.0848	0.0839	0.0832	0.0827	0.0825

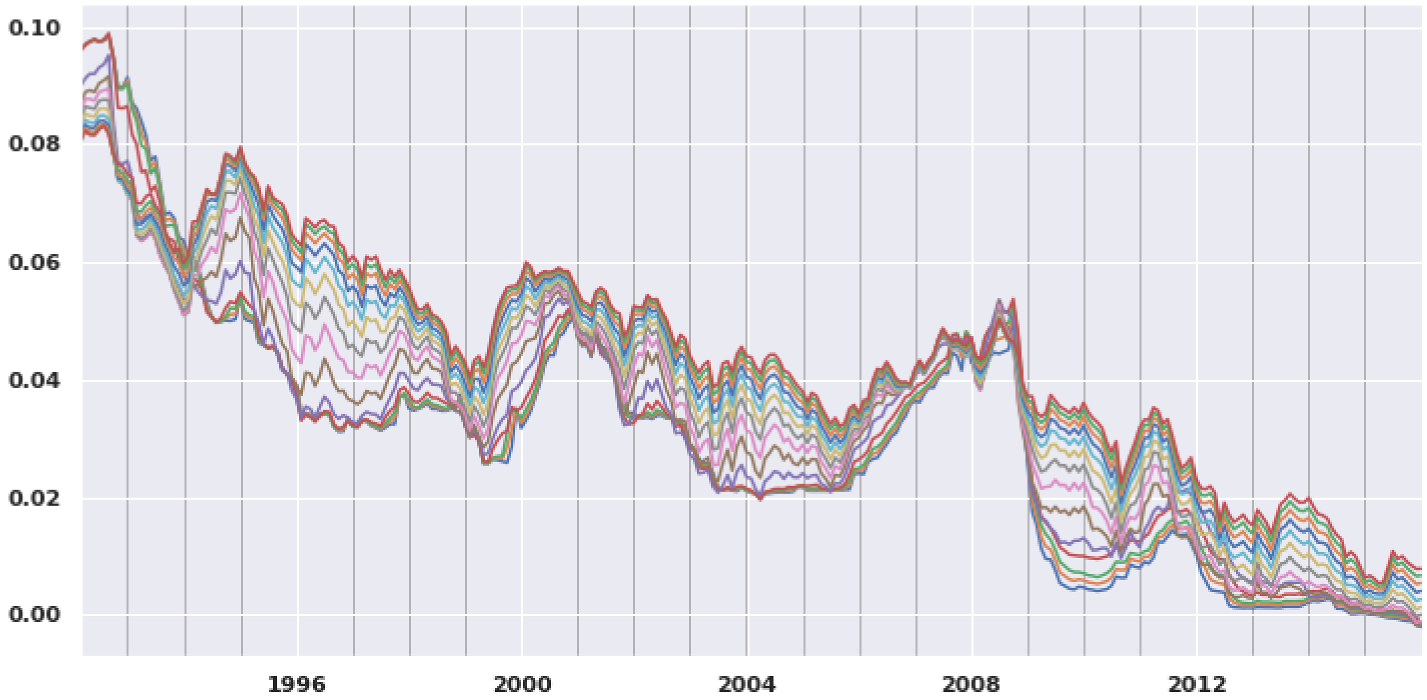
In [31]:
```
data_yc.shape
```
Out[31]:
```
(287, 14)
```

Each example (row) has 14 features: the yields for 14 maturity points on a given date.

Let's plot the history of yield curves

In [32]:
```
ych.plot_YC(data_yc)
```

```
/home/kjp/anaconda3/lib/python3.7/site-packages/pandas/plotting/_matplotlib/co
nverter.py:103: FutureWarning: Using an implicitly registered datetime convert
er for a matplotlib plotting method. The converter was registered by pandas on
import. Future versions of pandas will require you to explicitly register matp
lotlib converters.

To register the converters:
        >>> from pandas.plotting import register_matplotlib_converters
        >>> register_matplotlib_converters()
  warnings.warn(msg, FutureWarning)
```



Let's perform PCA on the **changes** in Yield Curve

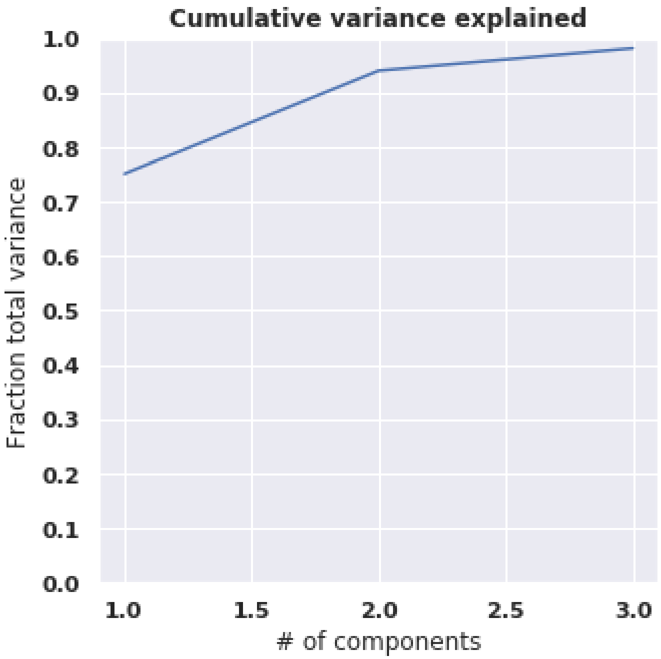
- Just like in Supervised Learning, we sometimes need to transform the data before fitting a model
- The features we feed to PCA are *yield changes* rather than yields

So $n = 14$ maturities, for m samples (many years of daily data)

How many bonds (i.e, what is the n') is "good enough" ?

The plot of cumulative variance explained, versus n' will give us an answer.

In [33]:
```
pca_yc, df_pca_yc = ych.doPCA(data_yc, doDiff=True)
_ = ych.plot_cum_variance(pca_yc)
```



Wow !

Only $n' = 3$ synthetic features capture almost all the variance of the original $n = 14$ features !

It gets even better !

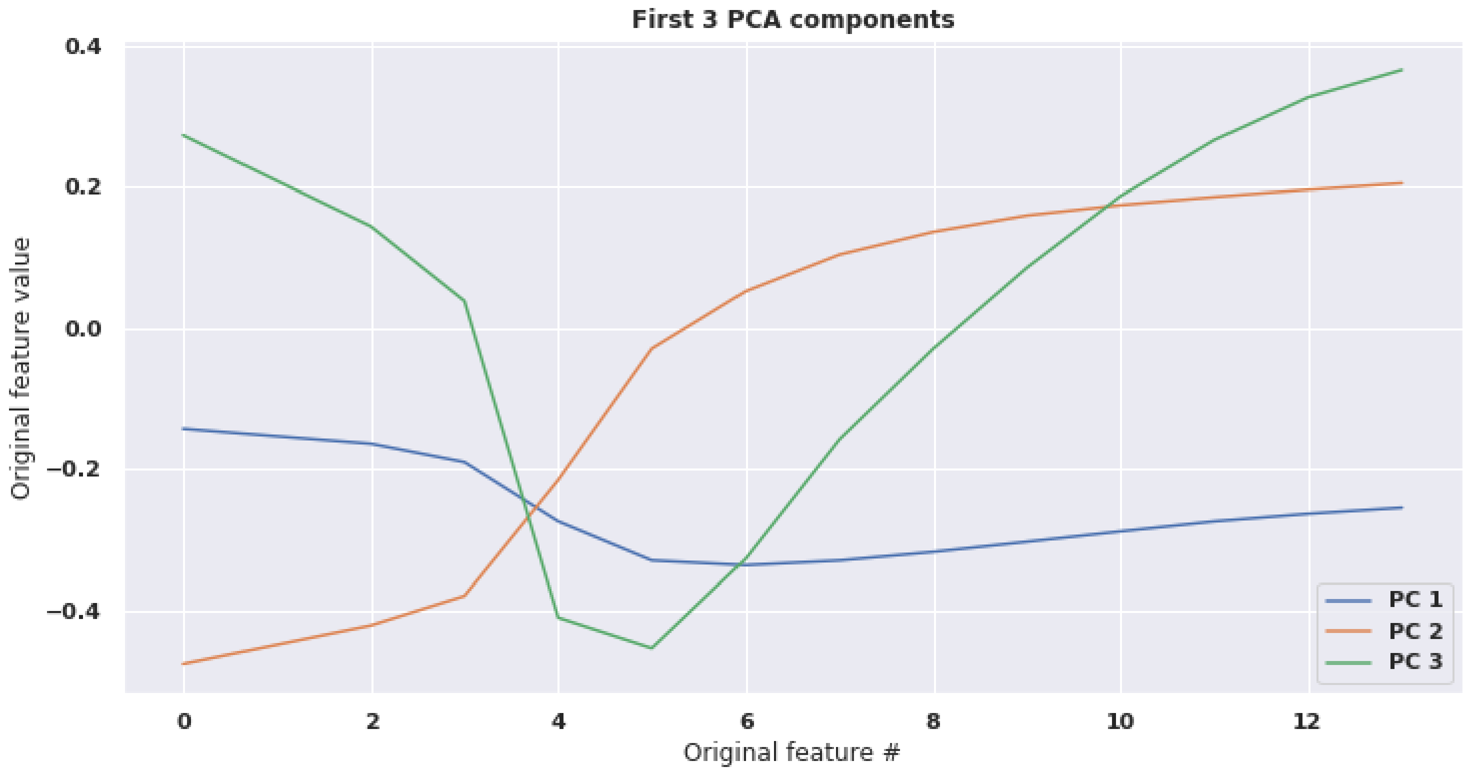
By examining the composition of the synthetic features, we can *interpret* what they are

Let's examine the effect of a 1 standard deviation move in each synthetic feature on original features.

We arrange the original features by maturity on the horizontal axis and plot the effect on the vertical.

The choice to arrange the horizontal axis by maturity is *very deliberate*, as we will see.

In [34]:
```
ych.plot_components(pca_yc)
```



This is a very typical pattern in Finance:

- the first synthetic feature affects all original features with roughly equal effect
- higher synthetic features often express a *dichotomy*
    - positive effect on some original features
    - negative effect on other original features

In our case the original features are yield changes.

A unit standard deviation value of synthetic feature j (PCj)

- $(j = 1)$: affects all original features (yield change) roughly equally
    - corresponds to a parallel shift in the Yield Curve
- $(j = 2)$: shows a dichotomy (of yield changes) between near and far maturities
    - corresponds to the slope of the Yield Curve changing
- $(j = 3)$: shows a dichotomy of yield changes of mid maturies versus extreme maturities
    - corresponds to a twist in the Yield Curve about the 5 year maturity

Recall

The synthetic features are standardized, hence 1 unit is one standard deviation.

The Σ matrix re-scales from standard deviation to original feature space.

- So can't compare the levels on the vertical axis between synthetic features.
  - Since $\sigma_1 > \sigma_2$
  - the absolute effect of synthetic feature 1 is greater than that of synthetic feature 2
  - for a 1 standard deviation move in each.

We started with a complicated sample (Yield Curve with $n = 14$ features)

- can almost completely explain (changes in the Yield Curve) with 3 intuitive market changes
    - Parallel Shift up/down of all maturities
    - Long end versus short end changes (slope)
    - Twist at intermediate maturity

That's interesting from a Machine Learning perspective but important for Finance:

- It shows show to construct efficient Hedge Portfolios.

Suppose we have a Target Portfolio consisting of long positions in many bonds, with many maturities.

Our goal is to construct another portfolio (the Hedge Portfolio)

- consisting of a small number of bonds
- with the *same* sensitivity to Yield Curve changes as the Target Portfolio

By combining a long position in the Target Portfolio with a short position in the Hedge Portfolio

- the resulting Net Portfolio is immunized (hedged) to changes in the Yield Curve

If we can re-express our Target Portfolio in terms of synthetic features

- We can construct the Hedging Portfolio as one consisting only of 3 synthetic bonds
- In quantities that mimic the sensitivity of the Target Portfolio to the first 3 PCs.
    - since the PCs are *independent*, this is easy
    - normally: you would need to solve
        - a system of 3 equations in 3 unknowns (size of position in each synthetic bond)

The catch is that each synthetic bond (feature) is a linear combination of $n = 14$ real bonds.

So we would need n real bonds to create each synthetic.

One way to deal with this is to project the synthetic features onto $n'' < 14$ real bonds.

So we come up with "approximate" versions of the synthetic features themselves.

The interpretation of Yield Curve changes guides us into a simpler hedge

- Hedge the parallel shift with the 10 year Treasury (most liquid bond in the US)
    - works because all yield changes roughly the same for first PC
- Hedge the slope with a long/short portfolio of 2 year/30 year (or 10 year) bonds
    - also liquid instruments, that are "close enough" to PC2
- Hedge the twist with a butterfly of long 2 year/10 year, short 5 year

So our Hedging Portfolio will consist of just a handful of very liquid bonds

- as opposed to n bonds, many of which may be illiquid

The primary way of hedging Fixed Income portfolios prior to this was by *duration* hedging

- assuming that yields across all maturities moved the same amount
- using a single liquid bond as the Hedge Portfolio

The PCA verified that the simple, intuitive hedge was actually the most important hedge to make !

Finance details

This section has little to do with Machine Learning but quite a bit to do with Fixed Income Finance.

- We have captured changes in *yield* of a bond
- To hedge *price* changes (our goal) we still need to translate a yield change to a price change

- For bond b with price P_b and yield y_b
  - we need $\frac{\partial P_b}{\partial y_b}$
    - the change in Price of bond b per unit change in its yield
  - this is known as the bond's *duration*
  - if we hold $\#_b$ units of bond b in a Portfolio
    - the bond's contribution to portfolio price change is $\#_b$ times the above sensitivity
  - sometimes more convenient to compute the *percent price* change per unit yield change
    - size of the hedge now in *number of dollars* rather than *number of bonds*

PCA of the SP 500

The same analysis that we did for the Bond Universe works for other instruments.

Consider a universe of all stocks in a particular stock universe.

We can perform PCA on the returns (percent changes) of each stock

- discover the common factors affecting all stocks in the universe.
- $n = 500$ features, for each example (one day return)

We don't have time to do it here, but the first components of many universes tends to be

- a first component that has roughly equal impact
    - PC1 is almost an equally weighted portfolio of all stocks
- higher components expressing dichotomies
    - cyclical stocks versus non-cyclical
    - large cap versus small cap
    - industry versus other industry

Interpretting the PC’s

The key in our interpetation of the PCs for the Yield Curve

- choice of ordering our original features by sorted maturity.

Had we chosen some other arrangement of the horizontal axis, we may not have seen the pattern.

So how do we find the "right" pattern ?

- Assign each original feature a set of attributes
    - bonds: maturity
    - stocks: industry, market capitization
- Propose a "theory" about how the value of an original feature will respond to a level of the PCj
    - The theory should relate the attribute of an original feature to the value of feature
- As a horizontal axis: sort by the attribute proposed by your theory
    - Stocks:
        - $(j = 1)$: arbitrary arrangement works, since all stocks repsonds equally
        - $(j > 1)$: cluster stocks by attribute
            - sort by market capitilization
            - group by industry: first all Industrials, then all Techs, etc.

Let $\hat{u}$ be the $1 \times n$ vector of all 0's except for a 1 at position i

$$\begin{aligned} \hat{u}_i &= 1 \\ \hat{u}_j &= 0, \quad j \neq i \end{aligned}$$

Then

$$s = \hat{u} I V^T$$

is a $1 \times n$ vector whose elements are the effect of a one standard deviation shock in synthetic factor i on each original feature.

$$s_j = \text{change in } X_j \text{ for a 1 standard deviation change in } \tilde{X}_i$$

$$s' = \hat{u} \Sigma V^T$$

is a $1 \times n$ vector that is scaled by the actual standard deviation of $\tilde{X}_i$ since the absolute size of a 1 standard deviation change in $\tilde{X}_i$ is σ_i.

That is: row i of V^T is the effect on each original feature of a one standard deviation shock in synthetic feature $\tilde{X}_i$.

Return to parent notebook

Recommender Systems: Pseudo SVD

There is another interesting use of Matrix Factorization that we will briefly review.

It will show both a case study and interesting extension of SVD.

Netflix Prize competition

- Predict user ratings for movies
- Dataset
    - Ratings assigned by users to movies: 1 to 5 stars
    - 480K users, 18K movies; 100MM ratings total
- $1MM prize
- awarded to team that beat Netflix existing prediction system by at least 10 percentage points

User preference matrix

We will try to use same language as PCA (examples, features, synthetic features)

- but map them to Netflix terms
    - Examples: Viewers
    - Features: Movies ("items")

Matrix $\mathbf{X}$: user rating of movies

$X_j^{(i)}$ is i^{th} user's rating of movie j

X is huge: $m * n$

- $m = .5$ million viewers
- $n = 18,000$ items (movies).

About 9 billion entries for a full matrix !

Idea: Linking Viewer to Movies via concepts

- Come up with your own "concepts" (synthetic features)
  - concept = attribute of a movie
    - map user preference to concept
    - map movie style to concept
    - supply and demand:
      - user demands concept, movie provides concept

Human defined concepts

- style: action, adventure, comedy, sci-fi
- actor
- typical audience segment

- Create user profile P: maps user to concept
- Create item profile Q: maps movies (features, items) to concept
- $\mathbf{X} = PQ^T$

One advantage of the $\mathbf{X} = PQ^T$ approach is a big space reduction.

With $k \leq n$ concepts:

- $\mathbf{X}$ is $m \times n$
- P is $m \times k$
- Q is $n \times k$

So idea is to factor X and discover $k \leq n$ synthetic features (concepts, "latent factors)

- $\mathbf{X} = PQ^T$
    - fit the model on training examples, i.e., perform decomposition
- Given a new user (test example), predict rating for an unseen movie:
    - Get a partial vector of user's ratings (defined for movies seen by user)
        - original features
    - Project onto concepts (synthetic features):
    - Inverse transform to get back complete vector in original feature space

SVD to discover concept

Why let a human guess what ML can discover ?

Use SVD to discover the k "best" synthetic features, rather than leaving it to a person.

Factor $\mathbf{X}$ by SVD !

$\mathbf{X} = U\Sigma * V^T$

Let $P = U\Sigma, Q = V$

Anyone spot the problem(s) ?

- First: a matrix with 9 billion entries is a handful !
- As you can imagine, any single user views only a fraction of the m movies
    - $\mathbf{X}$ is very sparse
    - Of the 9 billion potential entries in the full matrix $\mathbf{X}$, we only have 100 million defined.

How can we perform SVD on matrix with missing values ?

- Missing value imputation
    - not attractive: most values are missing
- Pseudo SVD

The ML mantra

- find a (cool ?) cost function that describes a solution to your problem
- Use Gradient Descent to solve

Pseudo SVD Loss function

The Froebenius Norm (used above) modified to exclude missing values

$$\mathcal{L}(\mathbf{X}', \mathbf{X}) = \sum_{\substack{1 \le i \le m, \\ 1 \le j \le n \\ \mathbf{X}_j^{(i)} \text{defined}}} \left(\mathbf{X}_j^{(i)} - \mathbf{X'}_j^{(i)} \right)^2$$

where $\mathbf{X}' = PQ^T$

Interpret as Reconstruction Error

Pseudo SVD algorithm

- Define $\mathbf{X}' = PQ^T$
- Take analytic derivatives of $Cost(X', X)$ with respect to
  - $P_j^{(i)}$ for $1 \leq i \leq m, 1 \leq j \leq k$
  - $Q_j^{(i)}$ for $1 \leq i \leq m, 1 \leq j \leq k$
- Initialize elements of P, Q randomly.
- Use Gradient Descent to solve for optimal entries of P, Q.
  - find entries of P, Q such that product matches non-empty part of $\mathbf{X}$

Note

- No guarantee that the P, Q obtained are
    - orthonormal, etc. (which SVD would give you)

But SVD won't work for $\mathbf{X}$ with missing values.

Filling in missing values

Once you have P, Q

- to predict a missing rating for user i movie j:

$$\hat{r}_{j,i} = q^{(\mathbf{i})} \cdot p_j^T$$

- $q^{(\mathbf{i})}$ is row i of Q
- p_j is column j of P^T

Some intuition

The rating vector of a user may have missing entries.

But we can still project to synthetic feature space based on the non-empty entries.

The projection winds up in a "neighborhood" of concepts.

Inverse transformation

- gets us to a completely non-empty rating vector that is a resident of this neighborhood.

Example

User rates

- Sci-Fi movies A and B very highly
- Does not rate Sci-Fi movie C.

Since A,B, C express same concept (Sci-Fi) they will be close in synthetic feature space.

Hence, the implied rating of User for movie C will be close to what other users rate C.

In [35]:

```
print("Done")
```

```
Done
```