Dimension Reduction -Principal Component Analysis (PCA)

Rina BUOY

AMERICAN UNIVERSITY OF PHNOM PENH

STUDY LOCALLY. LIVE GLOBALLY.

Agenda

1. **Introduction to Principal Component Analysis**

2. **Compute PCA**
    - Scikit Learn
    - Numpy SVD

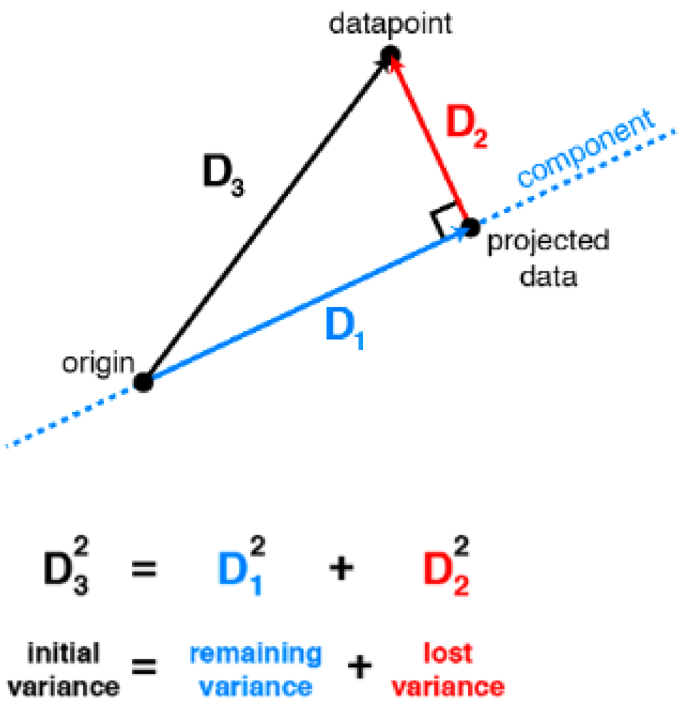
3. **Applications**
    - Visualization
    - Dimension Reduction
    - Image Compression

Why PCA?

- One of the more-useful methods from applied linear algebra
- Non-parametric way of extracting meaningful information from confusing data sets
- Uncovers hidden, low-dimensional structures that underlie your data
- These structures are more-easily visualized and are often interpretable to content experts

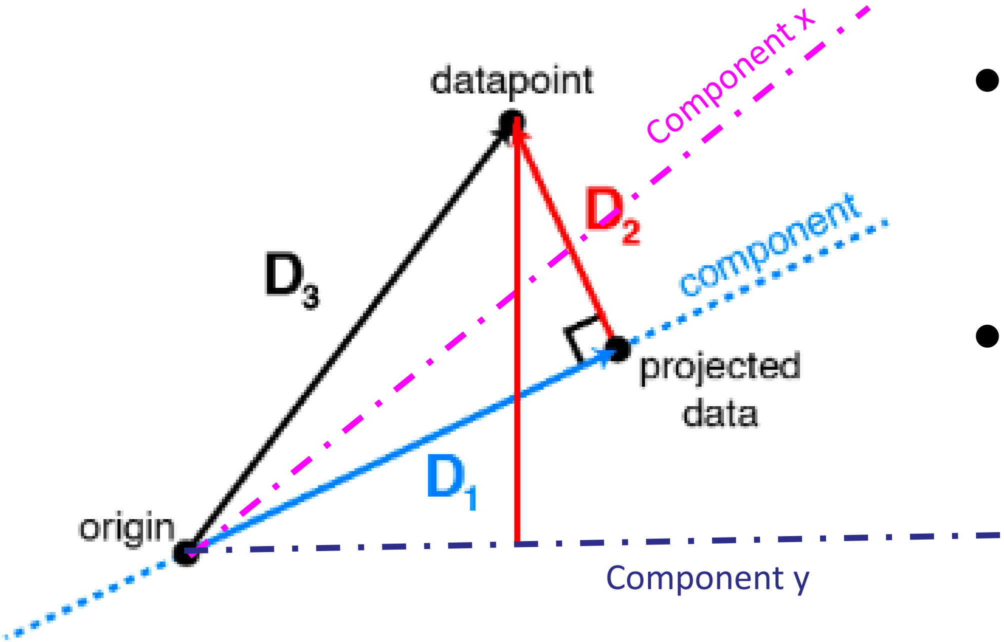
Source: DataCamp

Orthogonal Projection (1)



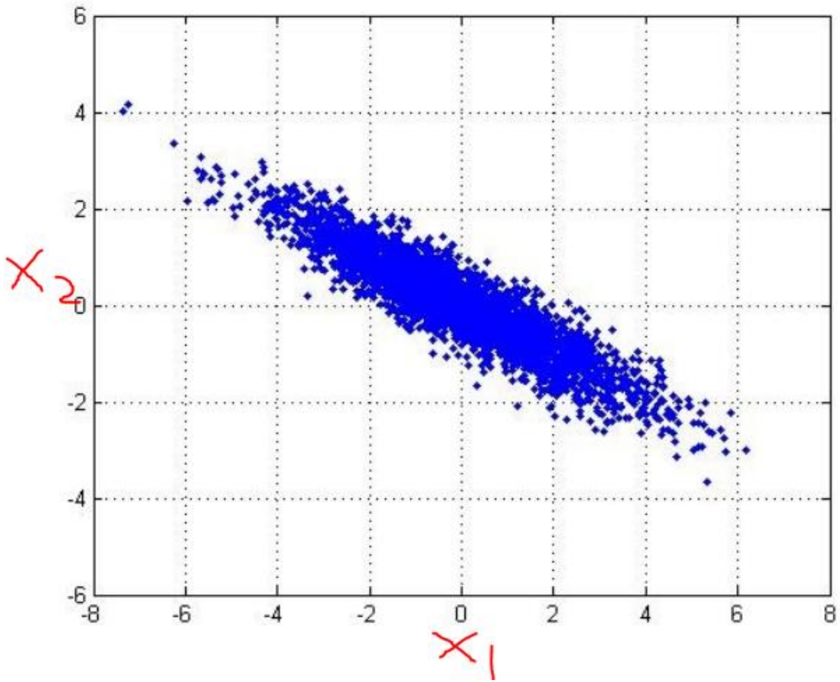
- The projection of ai onto the principal components relates the remaining variance to the squared residual by the Pythagorean theorem.
- Choosing the components to maximize variance is the same as choosing them to minimize the squared residuals.

Orthogonal Projection (2)



- Projection on ***component x*** has smaller lost variance than ***component*** and ***component y***.
- Projection on ***component y*** has higher lost variance than ***component*** and ***component x***.

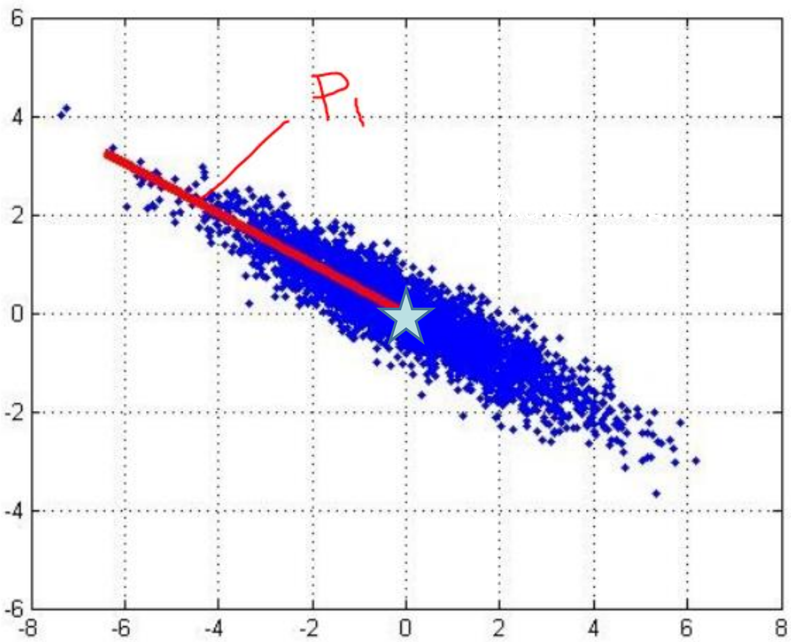
Computing Principal Components (1)



X1 & X2 are features of the dataset.

Which are the directions of the principal components for this dataset?

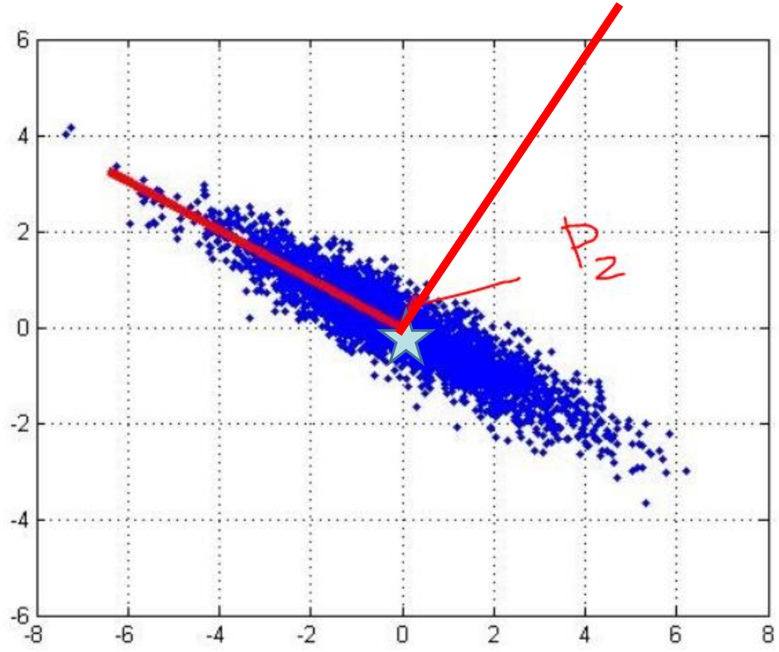
Computing Principal Components (2)



1. Start with the centroid (Xavg, Yavg)

1. Determine the direction along which sum of projection error /residual is minimised. We call this ‘first’ principal component.

Computing Principal Components (3)



3. The ‘second’ principle component is the direction which is orthogonal to the first one.

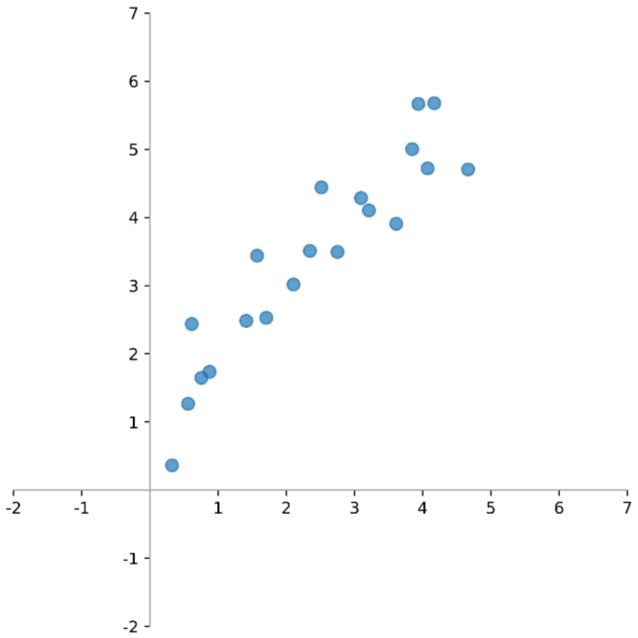
This is ***sequential*** algorithm.

Computing Principal Components (4)

- There are two other algorithms to compute PCA:
  1. Eigen Decomposition of Sample Covariance Matrix
  2. Singular Value Decomposition of Data Matrix (below section)

Computing PCA in Scikit Learn

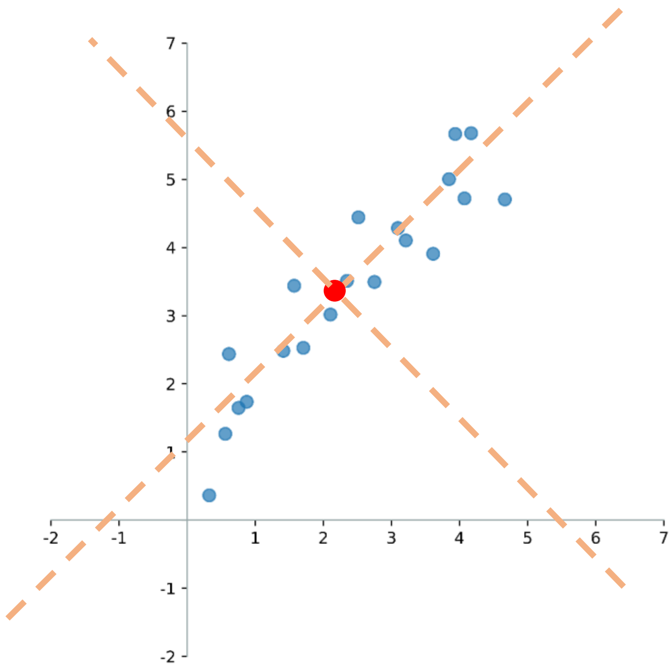
PCA in Scikit Learn (1)




```
from sklearn.decomposition import PCA
```

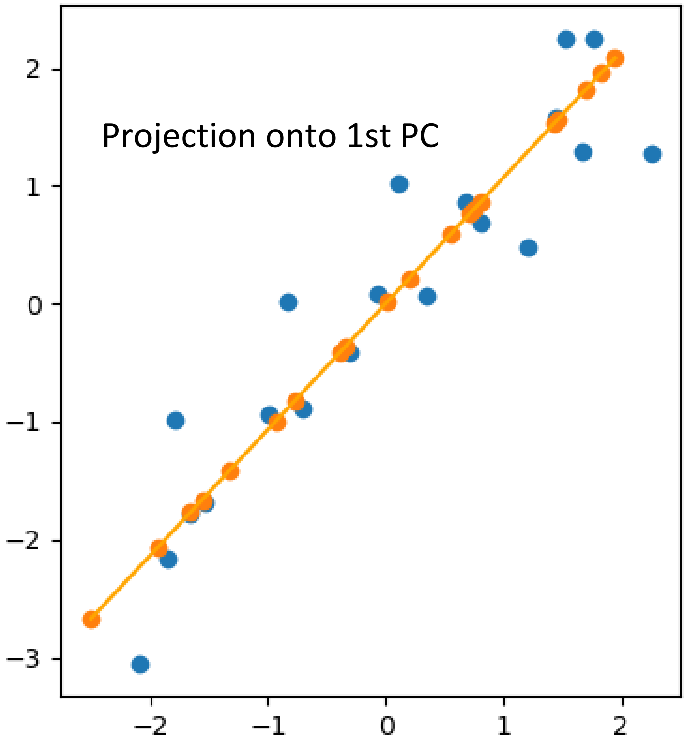
```
data = data = np.array([
 [1.41205819, 2.48438943],
 [0.87435703, 1.73657007],
 [3.84516794, 5.00362514],
 …
 [2.51401195, 4.44218097],
 [4.66614767, 4.70587771],
 [0.75369264, 1.64649525],
 [3.93658659, 5.66619787]
])
```

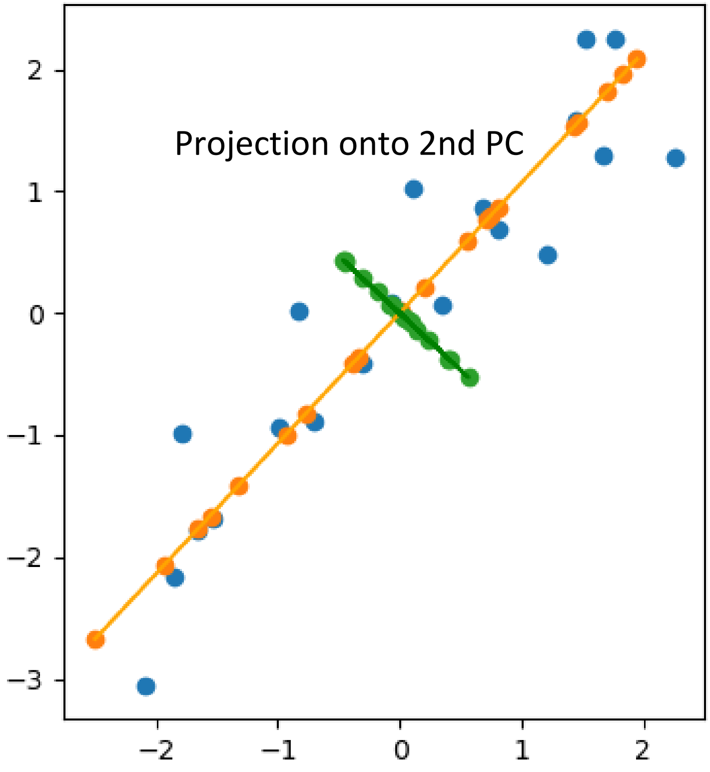
PCA in Scikit Learn (2)



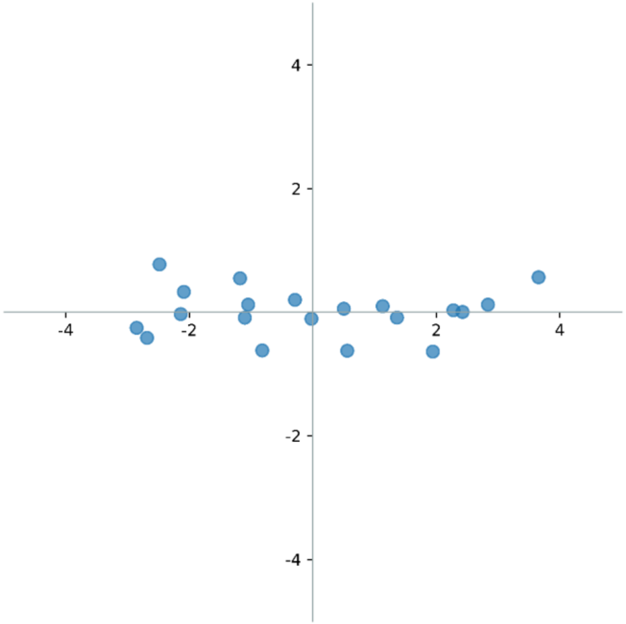

```
from sklearn.decomposition import PCA

data = data = np.array([
 [1.41205819, 2.48438943],
 [0.87435703, 1.73657007],
 [3.84516794, 5.00362514],
 …
 [2.51401195, 4.44218097],
 [4.66614767, 4.70587771],
 [0.75369264, 1.64649525],
 [3.93658659, 5.66619787]
])

pca = PCA(n_components = 2)
pca.fit(data)
```

PCA in Scikit Learn (3)





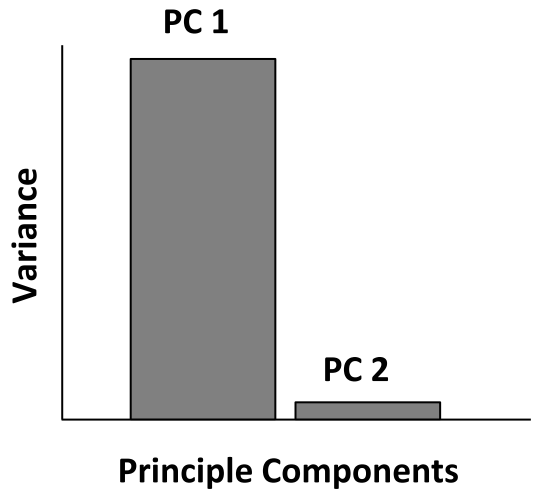
PCA in Scikit Learn (4)




```
from sklearn.decomposition import PCA

data = data = np.array([
 [1.41205819, 2.48438943],
 [0.87435703, 1.73657007],
 [3.84516794, 5.00362514],
 …
 [2.51401195, 4.44218097],
 [4.66614767, 4.70587771],
 [0.75369264, 1.64649525],
 [3.93658659, 5.66619787]
])

pca = PCA(n_components = 2)
pca.fit(data)

z = pca.transform(data)
```

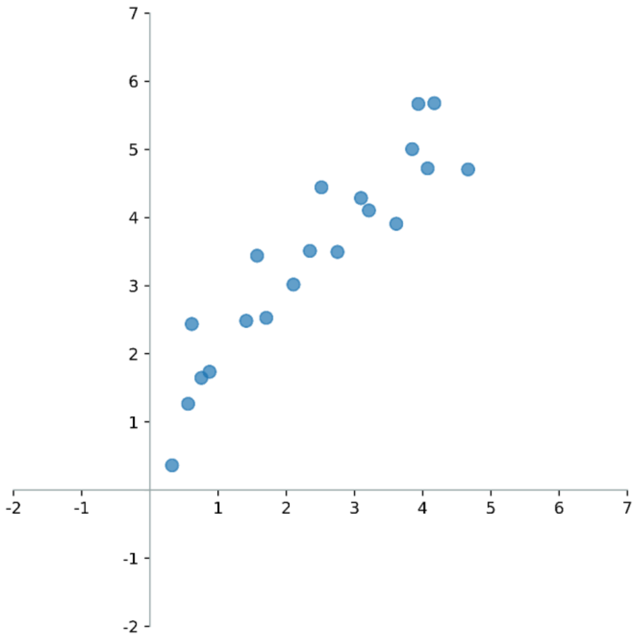
PCA in Scikit Learn (5)

```
features = range(pca._components_)

plt.bar(features,  pca.explained_variance_  )

plt.xticks(features)
plt.xlabel(“Principal Components”)
plt.ylabel(“Variance”)
```



- The higher the explained variance, the more important the component.
- Component with smaller explained variance can be dropped without losing significant information.

Numpy SVD




```python
import numpy as np

data = data = np.array([
 [1.41205819, 2.48438943],
 [0.87435703, 1.73657007],
 [3.84516794, 5.00362514],
 …
 [2.51401195, 4.44218097],
 [4.66614767, 4.70587771],
 [0.75369264, 1.64649525],
 [3.93658659, 5.66619787]
])
```

Numpy SVD

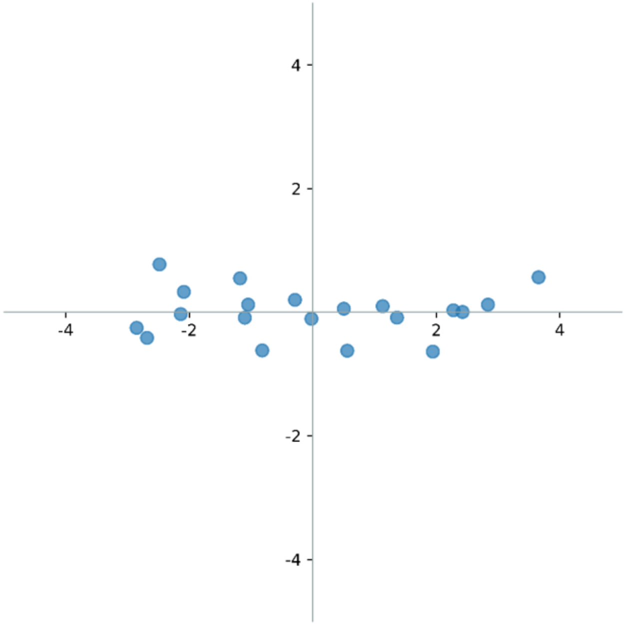
Data after transform

Principal components

$$data = U \times \Sigma \times V^T$$

```python
import numpy as np

 data = data = np.array([
  [1.41205819, 2.48438943],
  [0.87435703, 1.73657007],
  [3.84516794, 5.00362514],
  …
  [2.51401195, 4.44218097],
  [4.66614767, 4.70587771],
  [0.75369264, 1.64649525],
  [3.93658659, 5.66619787]
 ])

 u,s,vt = np.linalg.svd(data)
```

Numpy SVD



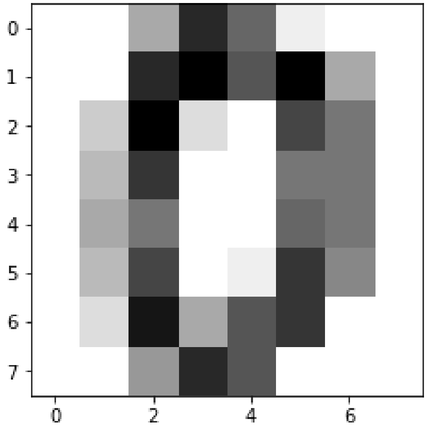

```
import numpy as np

 data = data = np.array([
  [1.41205819, 2.48438943],
  [0.87435703, 1.73657007],
  [3.84516794, 5.00362514],
  …
  [2.51401195, 4.44218097],
  [4.66614767, 4.70587771],
  [0.75369264, 1.64649525],
  [3.93658659, 5.66619787]
 ])

 u,s,vt = np.linalg.svd(data)


 z = np.dot(u, s)
```

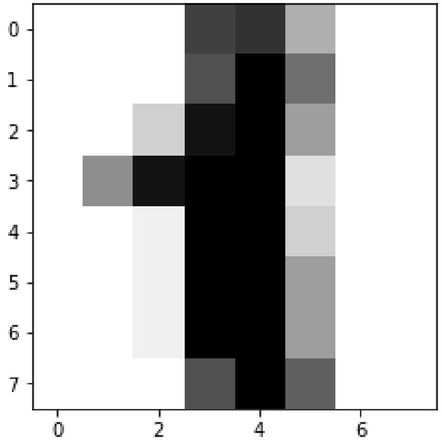
Applications

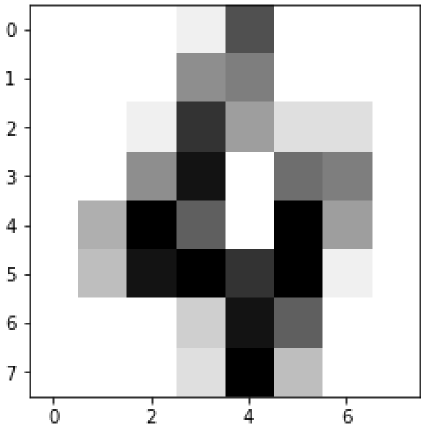
MNIST Dataset(1)

- The data set contains images of hand-written digits: 10 classes where each class refers to a digit.

- 1797 samples and 64 features representing 8x8 image of a digit







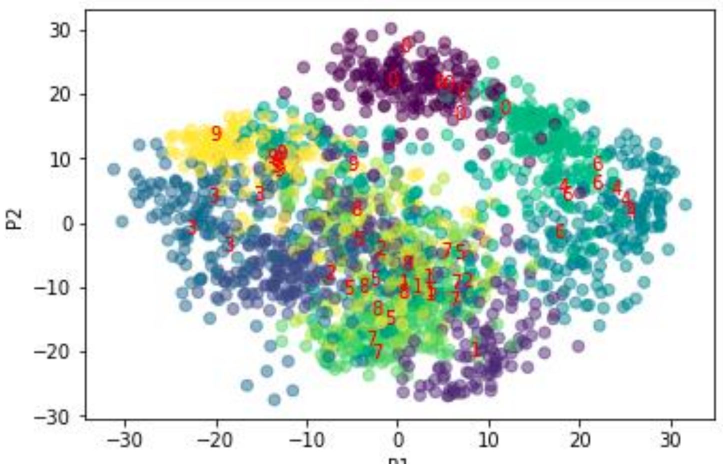
2D Visualization of High Dimension Dataset - MNIST (2)

In [71]:
```python
from sklearn.datasets import load_digits
digits= load_digits()
```

In [73]:
```python
print(digits.data.shape)
```

```
(1797, 64)
```

In [80]:
```python
pca = PCA(n_components=2)
pca_results = pca.fit_transform(digits.data)
plt.scatter(x=pca_results[:,0], y=pca_results[:,1], marker='o' , alpha=0.5,c=digits.target)
plt.xlabel('P1')
plt.ylabel('P2')
```



Dimension Reduction - MNIST (3)

- Training to recognize handwritten digit using Support Vector Machine (SVM)

- 64 features

- Score:

  0.98316

```python
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn import svm

digits = load_digits()

x_train, x_test, y_train, y_test = train_test_split(
  digits.data, digits.target,
  test_size=0.33, random_state=1
)

clf = svm.SVC(
  gamma='scale',
  decision_function_shape='ovo'
)

clf.fit(x_train, y_train)

clf.score(x_test, y_test)
```

Dimension Reduction - MNIST (4)

- Using PCA and reduce it to 10 features

- Score:

0.97474

```
from sklearn.decomposition import PCA

digits = load_digits()

pca = PCA(n_components = 10)
pca.fit(digits.data)
data = pca.transform(digits.data)

x_train, x_test, y_train, y_test = train_test_split(
  data, digits.target, test_size=0.33, random_state=1
)

clf = svm.SVC(
  gamma='scale',
  decision_function_shape='ovo'
)

clf.fit(x_train, y_train)
clf.score(x_test, y_test)
```

Image Compression

Image Compression (1)

400 pixels

600 pixels

100	0	20	10	30
100	20	10	50	20
200	30	30	40	50
244	222	100	50	60
67	78	20	24	33

A single pixel need 1 byte

This picture needs

$$600 \times 400 = 240{,}000 \text{ bytes}$$

Image Compression (2)

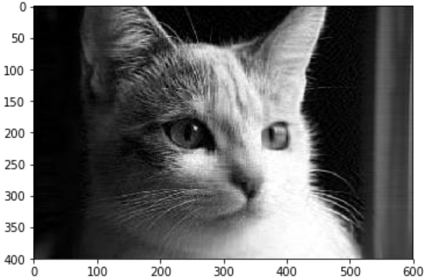
Original Picture

600×400

$=$ 600×400 $\times$ 400×400 $\times$ 400×400

Compressed picture

600×400



$=$ 600×50 $\times$ 50×50 $\times$ 50×400

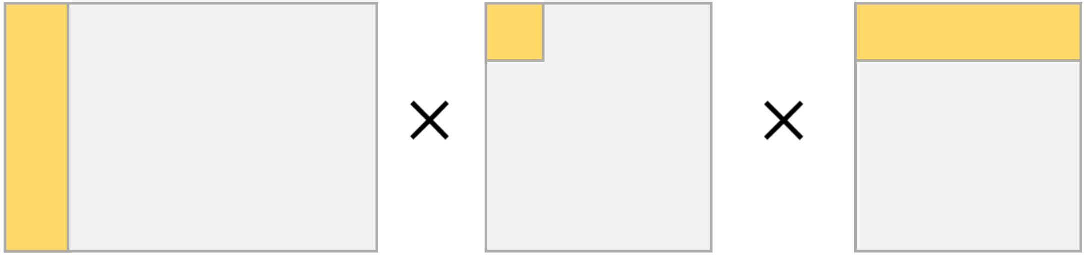
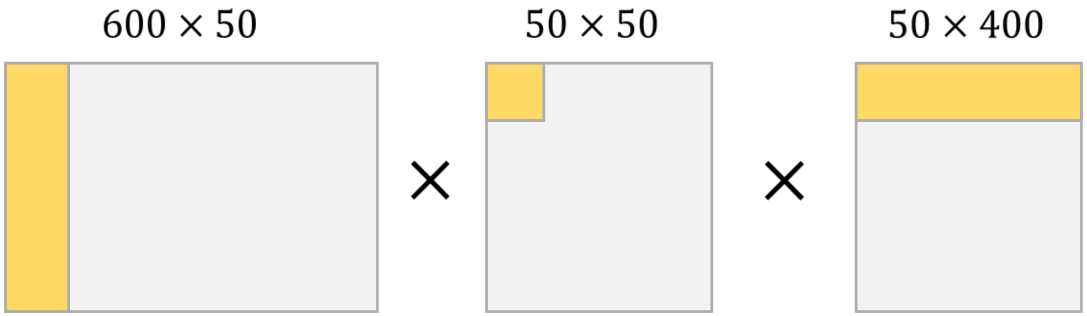


Image Compression (3)

Original Image

$600 \times 400 = 240{,}000$ bytes

Compressed image need



600 x 50 + 50 + 50 x 400 = 50,050 bytes

Image Compression (4)

```python
import matplotlib.pyplot as plt
import matplotlib.image as pimg
import numpy as np

img = pimg.imread(“cat.jpg”)
```

400 pixels

100	0	20	10	30	…	…
100	20	10	50	20	…	…
200	30	30	40	50	…	…
244	222	100	50	60	…	…
67	78	20	24	33	…	…
…	…	…	…	…	…	…
…	…	…	…	…	…	…

600 pixels

Image Compression (5)

```
import matplotlib.pyplot as plt
import matplotlib.image as pimg
import numpy as np


img = pimg.imread("cat.jpg")

u, s, vt = np.linalg.svd(img)
```

$$\overset{600 \times 400}{U} \times \overset{400 \times 400}{\Sigma} \times \overset{400 \times 400}{V^T}$$

Image Compression (6)

```python
import matplotlib.pyplot as plt
import matplotlib.image as pimg
import numpy as np


img = pimg.imread("cat.jpg")

u, s, vt = np.linalg.svd(img)

k = 50
ku = u[:,:k]
ks = np.diag(s[:k])
kvt = vt[:k]
kimg = np.dot(np.dot(ku,  ks), kvt)
```

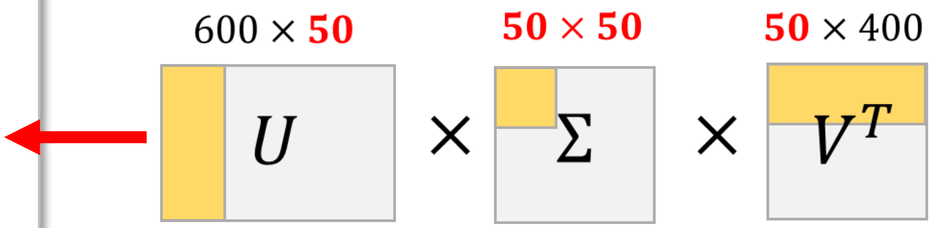


Image Compression (7)

```
import matplotlib.pyplot as plt
import matplotlib.image as pimg
import numpy as np

img = pimg.imread("cat.jpg")

u, s, vt = np.linalg.svd(img)

k = 50
ku = u[:,:k]
ks = np.diag(s[:k])
kvt = vt[:k]
kimg = np.dot(np.dot(ku,  ks), kvt)

plt.imshow(kimg, cmap='gray', vmin=0,
vmax=255)
```

Compressed picture

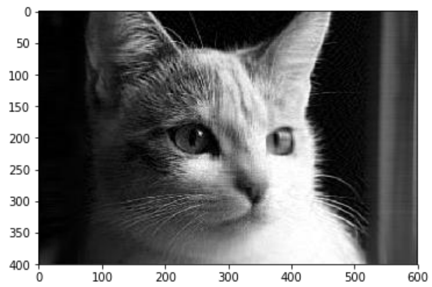
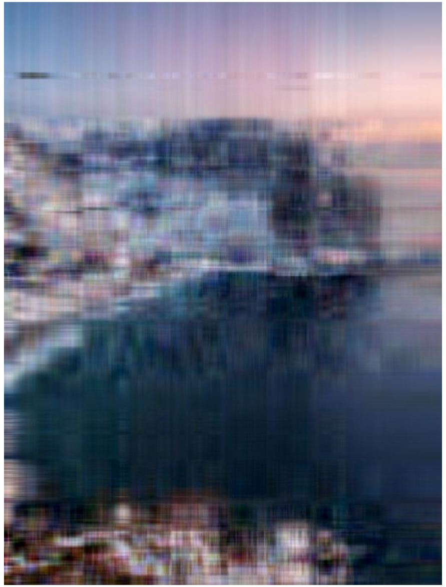
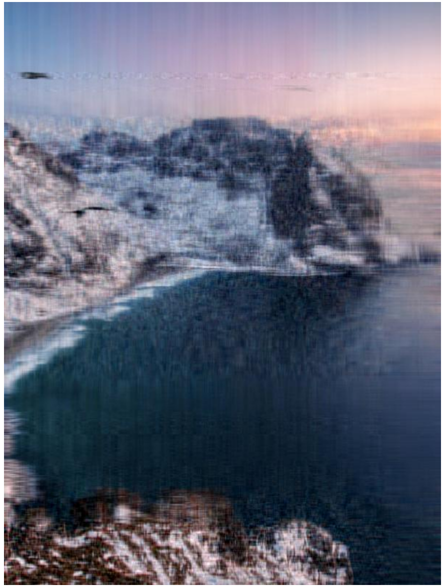


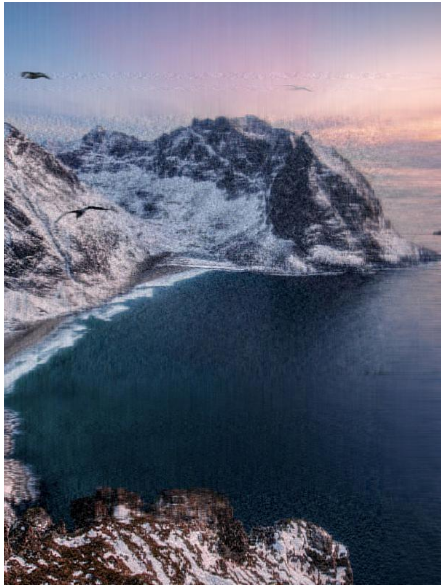
Image Compression with different k



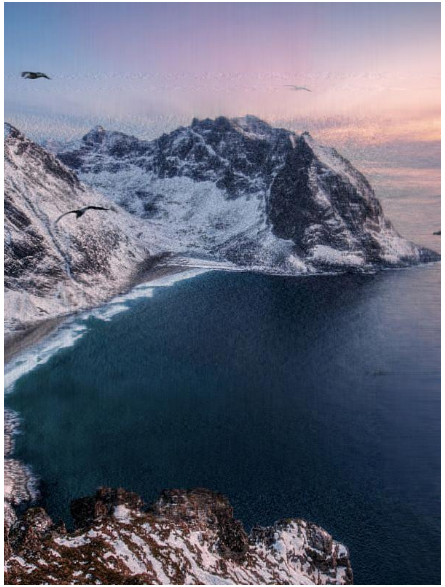
K: 10
Original size: **1,440,000 bytes**
Compressed size: **42,030 bytes**
Saved: **97.08125%**



K: 50
Original size: **1,440,000 bytes**
Compressed size: **210,150 bytes**
Saved: **85.40625%**



K: 70
Original size: **1,440,000 bytes**
Compressed size: **294,210 bytes**
Saved: **79.56875%**



K: 100
Original size: **1,440,000 bytes**
Compressed size: **420,300 bytes**
Saved: **70.8125%**