# Kernel PCA (Principal Component Analysis)

## Introduction:

Kernel Principal Component Analysis (Kernel PCA) is an extension of Principal Component Analysis (PCA) that uses kernel methods to implicitly map data into higher-dimensional spaces. It is particularly useful for capturing non-linear patterns in the data.

## Intuition:

The intuition behind Kernel PCA lies in transforming the input data into a higher-dimensional space using a kernel function (e.g., radial basis function - RBF kernel). In this higher-dimensional space, linear PCA is then applied to extract principal components. The use of a kernel allows capturing non-linear relationships between data points.

## Algorithm:

1. **Choose a Kernel Function:**
   - Common choices include the radial basis function (RBF) kernel, polynomial kernel, etc.
2. **Construct the Kernel Matrix:**
   - Compute the pairwise similarities between data points using the chosen kernel function.
3. **Center the Kernel Matrix:**
   - Center the kernel matrix to make the computation of principal components equivalent to the covariance matrix.
4. **Eigenvalue Decomposition:**
   - Perform eigenvalue decomposition on the centered kernel matrix to obtain eigenvalues and eigenvectors.
5. **Select Principal Components:**
   - Select the top k eigenvectors corresponding to the largest eigenvalues as the principal components.

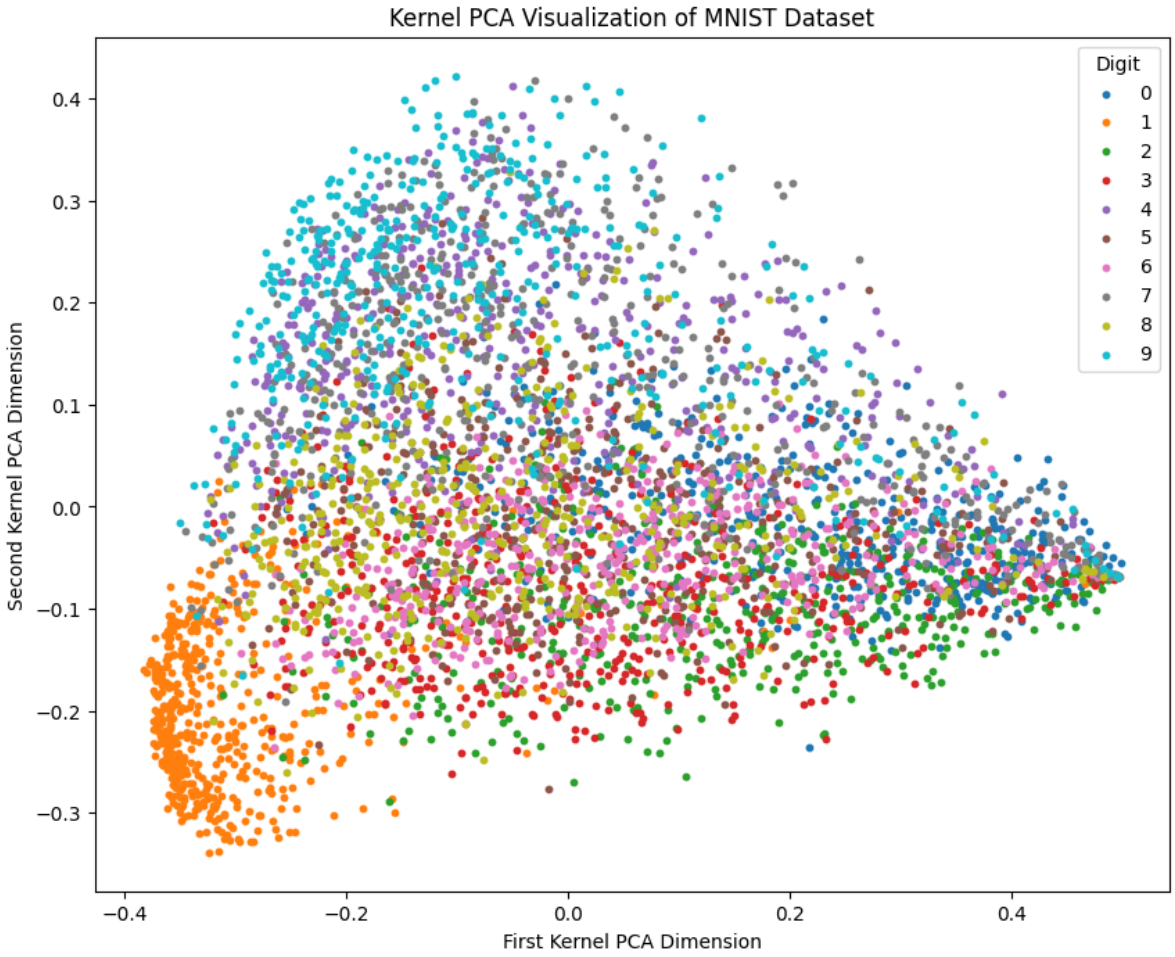
## Implementation in Python (MNIST Data):

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.decomposition import KernelPCA

# Load MNIST dataset
mnist = fetch_openml('mnist_784')
X = mnist.data.astype('float64')
y = mnist.target.astype('int')

# Sample only a subset of the data for faster processing (optional)
sample_size = 5000
idx = np.random.choice(len(X), sample_size, replace=False)
X_sample = X.iloc[idx]
y_sample = y.iloc[idx]
```

```
16
17  # Apply Kernel PCA to the data
18  n_components = 2
19  kpca = KernelPCA(n_components=n_components, kernel='rbf', gamma=0.05)
20  X_kpca = kpca.fit_transform(X_sample)
21
22  # Plot the results
23  plt.figure(figsize=(10, 8))
24  for digit in range(10):
25      plt.scatter(X_kpca[y_sample == digit, 0], X_kpca[y_sample == digit, 1],
    label=str(digit), s=10)
26  plt.title('Kernel PCA Visualization of MNIST Dataset')
27  plt.xlabel('First Kernel PCA Dimension')
28  plt.ylabel('Second Kernel PCA Dimension')
29  plt.legend(title='Digit', loc='upper right')
30  plt.show()
```

In this example:

- We use the `fetch_openml` function from scikit-learn to load the MNIST dataset.
- A subset of the dataset is sampled to speed up the processing, but you can adjust the `sample_size` based on your computational resources.
- Kernel PCA is applied to reduce the dimensionality to 2D using the radial basis function (RBF) kernel.
- The resulting Kernel PCA representation is visualized, and points are colored based on the digit they represent.



Feel free to experiment with different kernel functions (`linear`, `poly`, `rbf`, etc.) and parameters to observe how they affect the visualization. Adjust the `sample_size` based on your preferences.