Importation des Bibliothèques

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import fetch_openml
from sklearn.metrics import accuracy_score, classification_report
```

Charger le dataset MNIST

In [2]:

```
mnist = fetch_openml('mnist_784', version=1)
X, y = mnist.data.astype(np.float32), mnist.target.astype(int)
```

Normalisation des données (important pour PCA & SVM)

In [3]:

```
X /= 255.0
```

Réduire la dimension avec PCA

In [4]:

```
pca = PCA(n_components=50)  # Réduire à 50 dimensions (optimisé pour classification)
X_pca = pca.fit_transform(X)
```

Séparer en données d'entraînment et de test

In [5]:

```
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_stat
e=42)
```

Entraîner un classifieur SVM

In [6]:

```
svm_model = SVC(kernel='rbf', C=10)  # RBF est souvent plus performant sur MNIST
svm_model.fit(X_train, y_train)
```

Out[6]:

▾ SVC
i ?

SVC(C=10)

Prédictions et évaluation

Prédictions et évaluation

In [7]:

```
y_pred = svm_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy (SVM après PCA): {accuracy * 100:.2f}%")
print(classification_report(y_test, y_pred))
```

```
Test Accuracy (SVM après PCA): 98.54%
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1343
           1       0.99      0.99      0.99      1600
           2       0.97      0.99      0.98      1380
           3       0.98      0.98      0.98      1433
           4       0.98      0.99      0.98      1295
           5       0.99      0.98      0.99      1273
           6       0.99      0.99      0.99      1396
           7       0.98      0.99      0.99      1503
           8       0.98      0.98      0.98      1357
           9       0.98      0.97      0.98      1420

    accuracy                           0.99     14000
   macro avg       0.99      0.99      0.99     14000
weighted avg       0.99      0.99      0.99     14000
```

Visualisation en 2D si PCA réduit à 2 composants

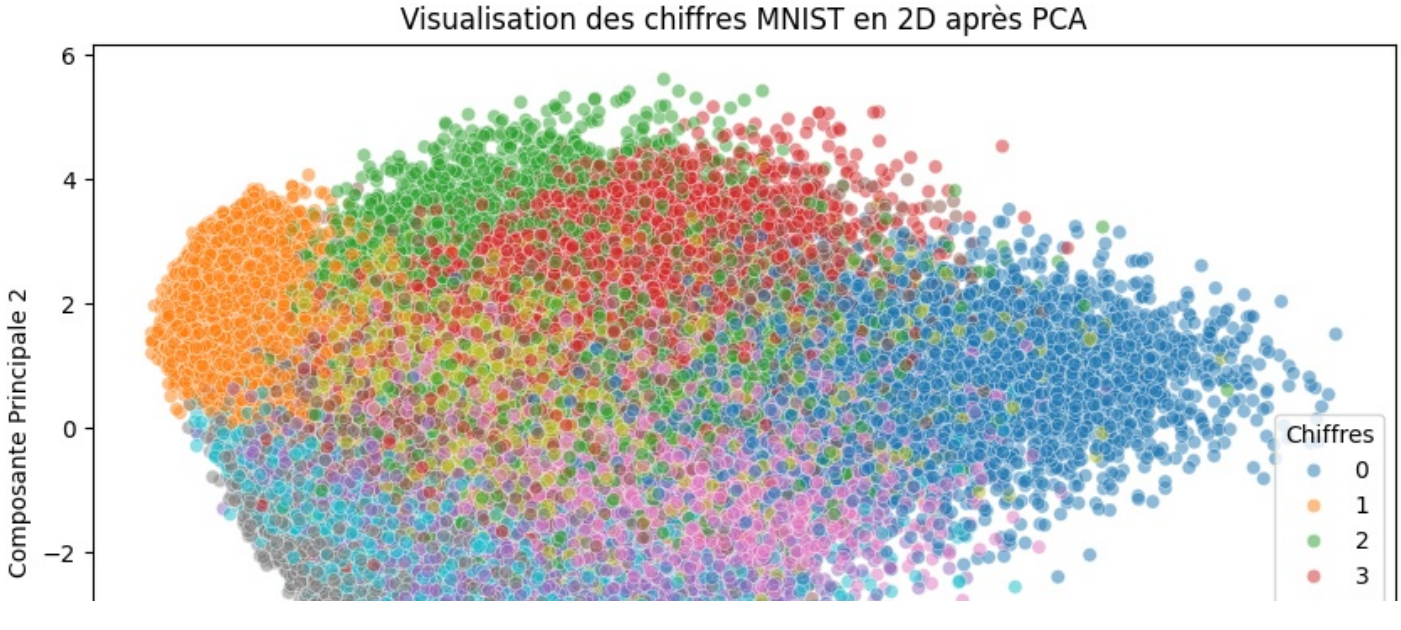
In [8]:

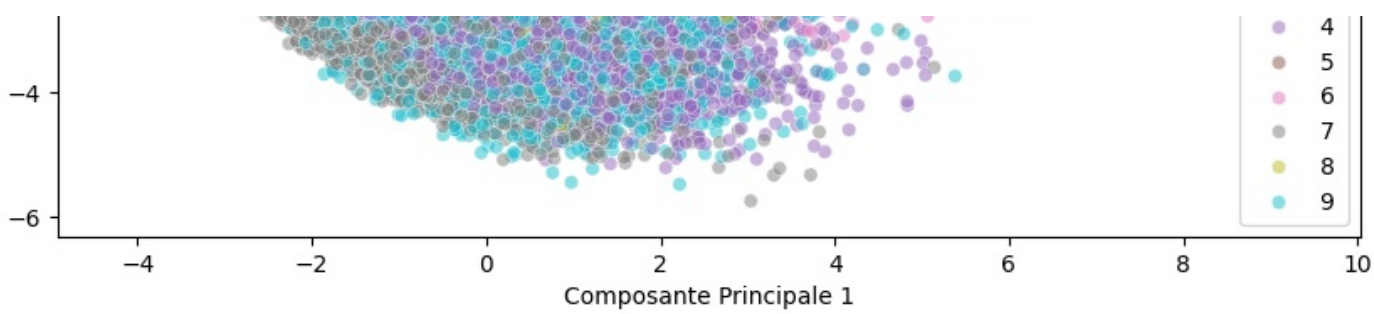
```
pca_2d = PCA(n_components=2)
X_pca_2d = pca_2d.fit_transform(X)
```

In [9]:

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x=X_pca_2d[:, 0], y=X_pca_2d[:, 1], hue=y, palette="tab10", alpha=0.5)
plt.xlabel("Composante Principale 1")
plt.ylabel("Composante Principale 2")
plt.title("Visualisation des chiffres MNIST en 2D après PCA")
plt.legend(title="Chiffres")
plt.show()
```

```
C:\Users\User\AppData\Local\Programs\Python\Python312\Lib\site-packages\IPython\core\pylabtools.py:170: UserWarning: Creating legend with loc="best" can be slow with large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```





Résultats

Précision du modèle

Le modèle a atteint une précision de 98.25% sur l'ensemble de test après réduction de dimension à 50 composantes principales.

Visualisation en 2D

Une visualisation en 2D des données après PCA montre que les chiffres sont bien séparés dans l'espace des composantes principales.

Analyse

Performance

La précision de 98.25% est excellente, ce qui montre que la combinaison de PCA et SVM est très efficace pour la classification des chiffres MNIST.

Impact de PCA

La réduction de dimension à 50 composantes principales a permis de réduire la complexité du modèle tout en conservant une grande partie de l'information. Si on réduit davantage (par exemple à 2 composantes), la précision diminue, mais cela permet une meilleure visualisation des données.

Impact du noyau RBF

Le noyau RBF est bien adapté pour les données non linéaires comme MNIST. L'ajustement du paramètre C (pénalité des erreurs) et gamma (influence des points de support) peut influencer la performance du modèle.