**Convertir en caractères**

train['label'] = train['label'].astype(str)

**Ou en float32**

# Conversion de la colonne 'label' en float32

train['label'] = train['label'].astype('float32')

**Retenir certaines colonnes**

df\_clean=df[['Class','Alcohol','Magnesium']]

**Combien de fois apparait tel ou tel valeurs**

train.label.value\_counts()

**Convertir en tableau Numpy**

La commande X = X.to\_numpy() convertit un objet pandas DataFrame en un tableau NumPy. Voici ce que chaque partie de la commande fait :

En résumé, cette commande est utilisée pour convertir un DataFrame pandas (X) en un tableau NumPy, ce qui peut être utile dans certains contextes, notamment si vous travaillez avec des bibliothèques qui utilisent NumPy pour les calculs numériques

**Chemin**

# Data path and file

#MNIST\_PATH= '../input/digit-recognizer'

#MNIST\_PATH= '../Another\_MNIST\_try/data/input/digit-recognizer'

CSV\_FILE\_TRAIN='train.csv'

CSV\_FILE\_TEST='test.csv'

def load\_mnist\_data(minist\_path, csv\_file):

csv\_path = os.path.join(minist\_path, csv\_file)

return pd.read\_csv(csv\_path)

def load\_mnist\_data\_manuel(minist\_path, csv\_file):

csv\_path = os.path.join(minist\_path, csv\_file)

csv\_file = open(csv\_path, 'r')

csv\_data = csv\_file.readlines()

csv\_file.close()

return csv\_data

def split\_train\_val(data, val\_ratio):

return

train = load\_mnist\_data(MNIST\_PATH,CSV\_FILE\_TRAIN)

test = load\_mnist\_data(MNIST\_PATH,CSV\_FILE\_TEST)

train\_2 = load\_mnist\_data\_manuel(MNIST\_PATH,CSV\_FILE\_TRAIN)

mnist\_label\_preds=cross\_val\_predict(KNeighborsClassifier(),mnist\_features\_prepared,mnist\_labels,cv=3,n\_jobs=-1)

**cross\_val\_predict**

cross\_val\_predict(KNeighborsClassifier(), mnist\_features\_prepared, mnist\_labels, cv=3, n\_jobs=-1)

cross\_val\_predict offers several advantages:

* **Insight into model performance**: By obtaining predictions for each data point, we can gain a deeper understanding of where the model works fine and where it struggles.
* **Data efficiency**: It ensures data efficiency as each data point is utilized for training and testing, which maximizes the dataset's use.
* **Effective evaluation**: We can assess the model's performance more accurately than a single train-test split

The main difference between predict and cross\_val\_predict is only trained on a single subset of the dataset, whereas cross\_val\_predict is trained and tested on all the data set in muliple intervals (depending on cv)

 cv=3: This specifies the number of folds for cross-validation. In this case, it's set to 3, meaning the dataset will be split into 3 folds for cross-validation.

 n\_jobs=-1: This parameter allows the algorithm to use all available CPU cores for parallel processing, which can speed up the cross-validation process.

n\_jobs=-1 signifie qu’on reprend tous les facteurs (on mettra ca par défaut)

**StandardScaler est important pour standardiser les données**

scaler=StandardScaler()

X\_data=scaler.fit\_transform(X)

**Chercher les différences**

**ACP - SVM**

L'ACP (Analyse en Composantes Principales) et le SVM (Machine à Vecteurs de Support) sont des techniques d'apprentissage automatique souvent utilisées dans des contextes différents. Cependant, il peut y avoir des scénarios où l'application de l'ACP avant le SVM peut être bénéfique. Voici quelques avantages potentiels :

1. **Réduction de la dimensionnalité :**
   * L'ACP est souvent utilisée pour réduire la dimensionnalité des données en projetant les variables originales dans un nouvel espace caractéristique basé sur les composantes principales. Cela peut être utile lorsque vous travaillez avec un grand nombre de caractéristiques.
   * En réduisant la dimensionnalité, vous pouvez potentiellement réduire la complexité du modèle SVM, améliorer le temps d'entraînement et réduire le risque de surajustement (overfitting) lorsque le nombre de caractéristiques est proche ou dépasse le nombre d'observations.
2. **Suppression du bruit et des corrélations :**
   * L'ACP peut aider à éliminer le bruit des données en se concentrant sur les composantes principales qui expliquent la plupart de la variance. Cela peut améliorer la capacité du SVM à généraliser sur de nouvelles données.
   * En éliminant les corrélations entre les variables, l'ACP peut également aider à éviter des problèmes potentiels liés à la multicollinéarité dans le cas où les variables sont fortement corrélées.
3. **Amélioration de la performance du modèle :**
   * En réduisant la dimensionnalité et en éliminant le bruit, l'ACP peut améliorer la performance globale du modèle SVM en facilitant l'identification de relations plus claires entre les caractéristiques et les étiquettes.
4. **Gestion des données hétérogènes :**
   * Si les caractéristiques des données sont mesurées dans différentes unités ou ont des échelles très différentes, l'ACP peut aider à homogénéiser les données en les transformant dans un espace où les dimensions sont sans unité.
   * Cela peut être bénéfique pour les SVM, qui peuvent être sensibles à l'échelle des caractéristiques.
5. **Interprétation facilitée :**
   * En réduisant le nombre de caractéristiques, l'ACP peut rendre le modèle SVM plus interprétable en focalisant l'attention sur les composantes principales les plus importantes.

**Ecrire dans un fichier**

with open('MNIST-test-set-predections.csv', 'wt') as ofile:

ofile.write("ImageId,Label\n")

**Recharger le noyau**

import os

os.\_exit(00)

**# Supprimer l'en-tête**

**# result = result.iloc[1:]**

**# Réinitialiser l'index**

**result = result.reset\_index(drop=True)**

**# Supprimer la colonne 'label'**

**result = result.drop('label', axis=1)**

**#Tableau de corrélation trié du plus petit grand (ou vice versa)**

corr\_matrix['label'].sort\_values(ascending=False)

**Creéere un chemin**

# Data path and file

#MNIST\_PATH= '../input/digit-recognizer'

#MNIST\_PATH= '../Another\_MNIST\_try/data/input/digit-recognizer'

CSV\_FILE\_TRAIN='train.csv'

CSV\_FILE\_TEST='test.csv'

def load\_mnist\_data(minist\_path, csv\_file):

csv\_path = os.path.join(minist\_path, csv\_file)

return pd.read\_csv(csv\_path)

def load\_mnist\_data\_manuel(minist\_path, csv\_file):

csv\_path = os.path.join(minist\_path, csv\_file)

csv\_file = open(csv\_path, 'r')

csv\_data = csv\_file.readlines()

csv\_file.close()

return csv\_data

def split\_train\_val(data, val\_ratio):

return

train = load\_mnist\_data(MNIST\_PATH,CSV\_FILE\_TRAIN)

test = load\_mnist\_data(MNIST\_PATH,CSV\_FILE\_TEST)

train\_2 = load\_mnist\_data\_manuel(MNIST\_PATH,CSV\_FILE\_TRAIN)

**Fonction zip :**

zip(cvres["mean\_test\_score"], cvres["params"]): This function pairs up elements from the mean test scores and parameters, creating an iterable of tuples.