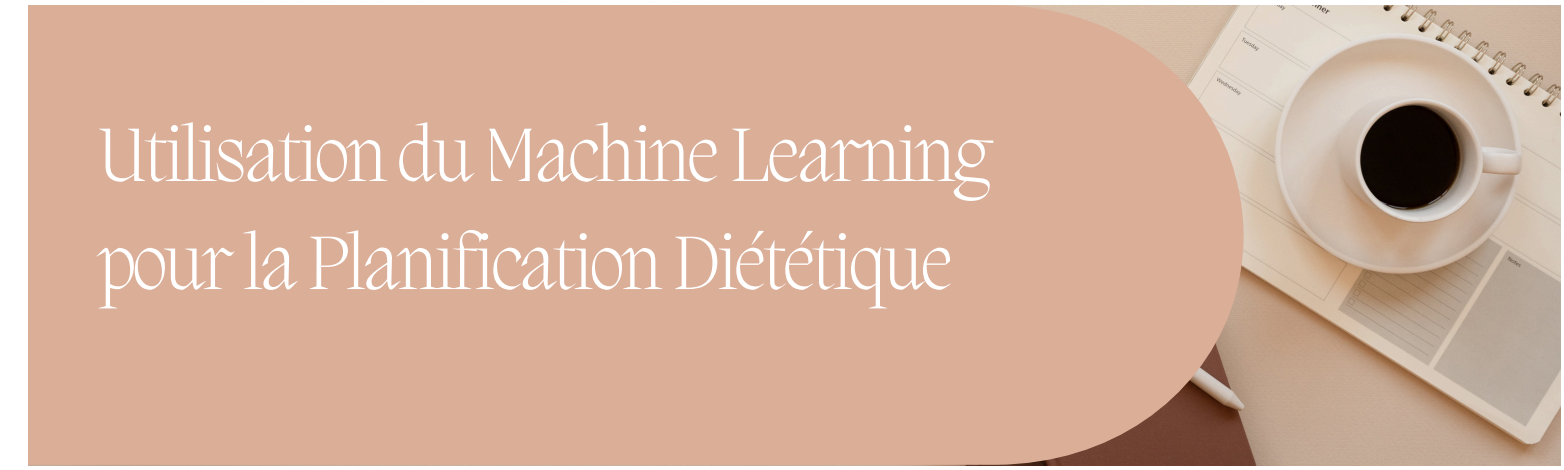
# Utilisation du Machine Learning pour la Planification Diététique




## Encadré par

**Mr. Hajja Zakaria**

## Réalisé par

**ELBOUZYANI Intissar**

**Hamdaoui Hajar**

**Jeddi Kawthar**

**Sommaire :**

# I- Introduction

**La nutrition joue un rôle crucial dans le maintien de la santé et du bien-être général. Avec l'augmentation des maladies liées à l'alimentation, il est essentiel de promouvoir des habitudes alimentaires saines. Le dataset "All_Diets.csv" propose une collection de recettes provenant de divers régimes alimentaires et cuisines, visant à fournir des options de repas saines et nutritives. Ce dataset, riche en informations nutritionnelles, offre une base précieuse pour l'analyse et l'optimisation des régimes alimentaires à l'aide des technologies de machine learning.**

# II- Objectif

L'objectif de ce projet est de développer un modèle de machine learning utilisant le dataset "All_Diets.csv" pour optimiser la planification diététique. Ce modèle permettra d'analyser les recettes en fonction de leur type de régime, de cuisine et de leur contenu nutritionnel (protéines, glucides et lipides). En utilisant ces données, nous visons à :

1. Prédire la valeur nutritionnelle des recettes.
2. Recommander des recettes adaptées à des régimes spécifiques.
3. Optimiser les plans alimentaires en fonction des besoins nutritionnels individuels.
4. Explorer les tendances et les corrélations entre différents types de régimes et de cuisines.

# II- Application

**1. Recommandations Personnalisées :**

Les modèles de machine learning peuvent analyser les besoins nutritionnels individuels et recommander des recettes adaptées à des régimes spécifiques, optimisant ainsi la nutrition quotidienne.

**2. Gestion des Maladies Chroniques :**

Pour les personnes souffrant de diabète, d'hypertension ou de maladies cardiaques, des plans alimentaires personnalisés peuvent aider à gérer ces conditions en contrôlant l'apport en macronutriments et en calories.

**3. Amélioration de la Performance Athlétique :**

Les athlètes peuvent bénéficier de recommandations diététiques personnalisées pour améliorer leur performance et leur récupération en fonction de leurs besoins énergétiques et de leurs objectifs d'entraînement.

**4. Promotion de la Santé Publique :**

Les recommandations diététiques basées sur les données peuvent être utilisées pour développer des programmes de santé publique visant à améliorer les habitudes alimentaires de la population générale.

**5. Assistance à la Perte de Poids :**

Des plans diététiques personnalisés peuvent aider les individus à atteindre leurs objectifs de perte de poids en fournissant des options de repas équilibrés et contrôlés en calories.

**6. Prévention des Carences Nutritionnelles :**

En analysant les habitudes alimentaires, les modèles de machine learning peuvent identifier les risques de carences nutritionnelles et suggérer des ajustements diététiques pour y remédier.

## IV- Dataset

- **Lien du Dataset**

https://www.kaggle.com/datasets/thedevastator/healthy-diet-recipes-a-comprehensive-dataset

- **Tableau des Colonnes et Leurs Descriptions**

| Column name | Description |
|---|---|
| **Diet_type** | The type of diet the recipe is for. (String) |
| **Recipe_name** | The name of the recipe. (String) |
| **Cuisine_type** | The cuisine the recipe is from. (String) |
| **Protein(g)** | The amount of protein in grams. (Float) |
| **Carbs(g)** | The amount of carbs in grams. (Float) |
| **Fat(g)** | The amount of fat in grams. (Float) |
| **Extraction_day** | The day the recipe was extracted. (String) |

- **Aperçu des Premières Lignes du Dataset**

```
   Diet_type                                        Recipe_name  \
0     paleo                      Bone Broth From 'Nom Nom Paleo'
1     paleo  Paleo Effect Asian-Glazed Pork Sides, A Sweet ...
2     paleo                                   Paleo Pumpkin Pie
3     paleo                        Strawberry Guacamole recipes
4     paleo  Asian Cauliflower Fried "Rice" From 'Nom Nom P...

       Cuisine_type  Protein(g)  Carbs(g)  Fat(g) Extraction_day  \
0          american        5.22      1.29    3.20     2022-10-16
1  south east asian      181.55     28.62  146.14     2022-10-16
2          american       30.91    302.59   96.76     2022-10-16
3           mexican        9.62     75.78   59.89     2022-10-16
4           chinese       39.84     54.08   71.55     2022-10-16
```

# V- Pre-Processing

1. **Sélection des Caractéristiques Pertinentes** :

- Dans cette étape, on a identifié les caractéristiques les plus pertinentes pour notre analyse. On a choisi 'Recipe_name', 'Cuisine', 'Protein(g)', 'Carbs(g)', 'Fat(g)' comme caractéristiques à inclure dans Notre ensemble de données X, car elles sont susceptibles d'avoir un impact sur le type de régime alimentaire ('Diet_type').

```
[ ]  1 # Prétraiter les données
     2 X = df[['Recipe_name', 'Cuisine', 'Protein(g)', 'Carbs(g)', 'Fat(g)']]
     3 y = df['Diet_type']
     4
```

2. **Encodage One-Hot des Étiquettes** :

- On a utilisé la technique de l'encodage one-hot sur les étiquettes 'Diet_type' pour transformer les étiquettes catégorielles en vecteurs binaires. Cela crée une représentation appropriée pour les algorithmes d'apprentissage automatique, où chaque étiquette catégorielle devient une série de 0 et de 1.

```
[ ]  1 # Effectuer l'encodage One-Hot Encoding sur les étiquettes
     2 y_encoded = pd.get_dummies(y)
```

3. **Identification des Caractéristiques Numériques et Catégorielles** :

- On a séparé mes caractéristiques en deux catégories : numériques et catégorielles. Les caractéristiques numériques sont 'Protein(g)', 'Carbs(g)', et 'Fat(g)', tandis que les caractéristiques catégorielles incluent 'Diet_type', 'Recipe_name', et 'Cuisine_type'.

```
[ ]  1 # Identification des colonnes numériques et catégorielles
     2 numeric_features = ['Protein(g)', 'Carbs(g)', 'Fat(g)']
     3 categorical_features = ['Diet_type', 'Recipe_name', 'Cuisine_type']
```

4. **Mise à l'Échelle des Données avec StandardScaler** :

- Pour garantir que les caractéristiques numériques ont des échelles similaires et pour éviter toute dominance d'une caractéristique sur une autre, on a utilisé le StandardScaler pour mettre à l'échelle les caractéristiques numériques ('Protein(g)', 'Carbs(g)', 'Fat(g)'). Cela rend les données plus adaptées à la modélisation et à l'apprentissage automatique.

```
1 # Mettre à l'échelle les données en utilisant StandardScaler
2 scaler = StandardScaler()
3 X_scaled = scaler.fit_transform(X)
```

# VI- Modèles

### I- Premier Modèle utilisé: réseau de neurones

**Montage du Drive Google, commandes Bash et importation de Bibliothèques Python:**

```
[ ]  1 from google.colab import drive
     2 drive.mount("/content/drive")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[ ]  1 from google.colab import drive
     2 drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[ ]  1 !pwd
```

```
/content
```

```
[ ]  1 !ls
```

```
drive  sample_data
```

```
[ ]  1 !ls /content/drive/MyDrive/All_Diets.csv
     2
```

```
/content/drive/MyDrive/All_Diets.csv
```

```
[ ]  1 import pandas as pd
     2 from sklearn.preprocessing import StandardScaler
     3 from sklearn.model_selection import train_test_split
     4 from tensorflow.keras.models import Sequential
     5 from tensorflow.keras.layers import Dense
     6 from tensorflow.keras.optimizers import Adam
     7 from tensorflow.keras.utils import to_categorical
     8 from tensorflow.keras.callbacks import ReduceLROnPlateau
     9 from sklearn.metrics import accuracy_score, mean_squared_error, mean_absolute_error
    10 import numpy as np
```

1. **La premiére partie de ce code concerne la division des données en ensembles d'entraînement et de Test**

- Utilise la fonction train_test_split pour diviser les données préalablement mises à l'échelle X_scaled en ensembles d'entraînement et de test, ainsi que les étiquettes encodées y_encoded. Les données de test représentent 20% de l'ensemble des données, et le paramètre random_state=42 assure la reproductibilité des résultats.

2. **Et puis on a crée un modèle de Réseau de Neurones** :
   - Utilise la classe Sequential pour construire un modèle de réseau de neurones avec plusieurs couches. Le modèle comprend des couches denses avec des fonctions d'activation LeakyReLU, des couches de normalisation par lots (BatchNormalization) pour améliorer la stabilité et des couches de dropout pour prévenir le surapprentissage.
3. **Compilation du Modèle** :
   - Utilise l'optimiseur Adam avec un taux d'apprentissage de 0.001 pour compiler le modèle. La fonction de perte est définie comme la perte de catégorisation croisée catégorielle et la métrique d'évaluation est l'exactitude (accuracy).
4. **Planification du Taux d'Apprentissage** :
   - Un planificateur de taux d'apprentissage est utilisé pour ajuster dynamiquement le taux d'apprentissage pendant l'entraînement en fonction de la performance du modèle sur les données de validation. Cela peut aider à accélérer ou à ralentir l'apprentissage en fonction de la situation.
5. **Entraînement du Modèle** :
   - Le modèle est entraîné sur les données d'entraînement et de validation en utilisant la méthode fit(). Pendant l'entraînement, le modèle ajuste ses poids pour minimiser la perte définie et maximiser la précision sur les données fournies.

```
[ ]
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded, test_size=0.2, random_state=42)
```

```
# Créer le modèle de réseau de neurones
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization, LeakyReLU

# Créer le modèle de réseau de neurones
model = Sequential()
model.add(Dense(128, input_dim=3))
model.add(BatchNormalization())
model.add(LeakyReLU(alpha=0.1))
model.add(Dropout(0.3))
model.add(Dense(64))
model.add(BatchNormalization())
model.add(LeakyReLU(alpha=0.1))
model.add(Dropout(0.3))
model.add(Dense(32))
model.add(BatchNormalization())
model.add(LeakyReLU(alpha=0.1))
model.add(Dropout(0.3))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(5, activation='softmax'))

# Compiler le modèle avec optimizations
optimizer = Adam(learning_rate=0.001)
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])

# Learning Rate Scheduler
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.0001)


```

```
[ ]
# Entraîner le modèle avec les données d'entraînement et de test aléatoires
model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test))
```

**Output ==>**

```
Epoch 1/100
196/196 [==============================] - 3s 6ms/step - loss: 1.5110 - accuracy: 0.3376 - val_loss: 1.4570 - val_accuracy: 0.4129
Epoch 2/100
196/196 [==============================] - 1s 4ms/step - loss: 1.4496 - accuracy: 0.3767 - val_loss: 1.3931 - val_accuracy: 0.4161
Epoch 3/100
196/196 [==============================] - 1s 5ms/step - loss: 1.4283 - accuracy: 0.3865 - val_loss: 1.3579 - val_accuracy: 0.4245
Epoch 4/100
196/196 [==============================] - 1s 4ms/step - loss: 1.4106 - accuracy: 0.3973 - val_loss: 1.3557 - val_accuracy: 0.4264
Epoch 5/100
196/196 [==============================] - 1s 3ms/step - loss: 1.4111 - accuracy: 0.4028 - val_loss: 1.3577 - val_accuracy: 0.4181
Epoch 6/100
196/196 [==============================] - 1s 4ms/step - loss: 1.4062 - accuracy: 0.4020 - val_loss: 1.3523 - val_accuracy: 0.4277
Epoch 7/100
196/196 [==============================] - 1s 4ms/step - loss: 1.3966 - accuracy: 0.4097 - val_loss: 1.3482 - val_accuracy: 0.4277
Epoch 8/100
196/196 [==============================] - 1s 5ms/step - loss: 1.3952 - accuracy: 0.4140 - val_loss: 1.3493 - val_accuracy: 0.4270
Epoch 9/100
196/196 [==============================] - 1s 5ms/step - loss: 1.3915 - accuracy: 0.4142 - val_loss: 1.3579 - val_accuracy: 0.4264
Epoch 10/100
196/196 [==============================] - 1s 5ms/step - loss: 1.3904 - accuracy: 0.4110 - val_loss: 1.3448 - val_accuracy: 0.4270
Epoch 11/100
196/196 [==============================] - 1s 5ms/step - loss: 1.3887 - accuracy: 0.4122 - val_loss: 1.3517 - val_accuracy: 0.4149
Epoch 12/100
196/196 [==============================] - 1s 5ms/step - loss: 1.3891 - accuracy: 0.4132 - val_loss: 1.3422 - val_accuracy: 0.4149
Epoch 13/100
196/196 [==============================] - 1s 3ms/step - loss: 1.3893 - accuracy: 0.4174 - val_loss: 1.3432 - val_accuracy: 0.4245
Epoch 14/100
196/196 [==============================] - 1s 3ms/step - loss: 1.3830 - accuracy: 0.4185 - val_loss: 1.3414 - val_accuracy: 0.4181
Epoch 15/100
196/196 [==============================] - 1s 4ms/step - loss: 1.3790 - accuracy: 0.4196 - val_loss: 1.3354 - val_accuracy: 0.4213
Epoch 16/100
196/196 [==============================] - 1s 3ms/step - loss: 1.3820 - accuracy: 0.4138 - val_loss: 1.3330 - val_accuracy: 0.4302
Epoch 17/100
196/196 [==============================] - 1s 3ms/step - loss: 1.3762 - accuracy: 0.4231 - val_loss: 1.3368 - val_accuracy: 0.4123
Epoch 18/100
196/196 [==============================] - 1s 3ms/step - loss: 1.3816 - accuracy: 0.4201 - val_loss: 1.3413 - val_accuracy: 0.4200
Epoch 19/100
196/196 [==============================] - 1s 4ms/step - loss: 1.3758 - accuracy: 0.4178 - val_loss: 1.3393 - val_accuracy: 0.4264
```

**Dans cette partie du code:**

1. **Évaluation du Modèle** :
   - La méthode evaluate() est utilisée pour évaluer le modèle sur l'ensemble de test. Cela calcule la perte et la précision du modèle sur les données de test.
2. **Analyse des Erreurs** :
   - Les prédictions du modèle sont obtenues pour l'ensemble de test à l'aide de model.predict(X_test).
   - Les indices des prédictions incorrectes sont identifiés en comparant les prédictions avec les étiquettes réelles.
3. **Analyse des Indices d'Erreurs Uniques** :
   - Les indices des prédictions incorrectes sont convertis en un tableau numpy.
   - Ensuite, les indices uniques des prédictions incorrectes sont obtenus à l'aide de la fonction np.unique().

```
[ ]  1 # Évaluer le modèle
     2 loss, accuracy = model.evaluate(X_test, y_test)
     3 print(f"Perte : {loss:.2f}")
     4 print(f"Précision : {accuracy:.2f}")
```

```
49/49 [==============================] - 0s 3ms/step - loss: 1.3425 - accuracy: 0.4245
Perte : 1.34
Précision : 0.42
```

```
[ ]  1 # Analyse des erreurs
     2 y_pred = model.predict(X_test)
     3 errors = np.where(y_pred != y_test)[0]
     4 print("Indices des prédictions incorrectes :", errors)
     5
     6
```

```
49/49 [==============================] - 0s 2ms/step
Indices des prédictions incorrectes : [   0    0    0 ... 1561 1561 1561]
```

```
[ ]  1 # Convertir les indices des prédictions incorrectes en un tableau numpy
     2 errors = np.array(errors)
     3
     4 # Obtenir les indices uniques des prédictions incorrectes
     5 unique_errors = np.unique(errors)
     6
     7 # Afficher les indices uniques des prédictions incorrectes
     8 print("Indices uniques des prédictions incorrectes :", unique_errors)
     9
```

```
Indices uniques des prédictions incorrectes : [   0    1    2 ... 1559 1560 1561]
```

1. **Analyse des Erreurs de Prédiction du Modèle**

- Ce bloc de code convertit les ensembles de test X_test et y_test en DataFrame pandas pour une meilleure visualisation et analyse. Ensuite, il réinitialise les index des DataFrames pour correspondre aux indices des prédictions incorrectes. Enfin, il affiche les caractéristiques, les étiquettes réelles et les prédictions du modèle pour les exemples correspondant aux indices uniques des prédictions incorrectes.

```python
# Convertir X_test et y_test en DataFrame pandas
X_test_df = pd.DataFrame(X_test, columns=['Protein(g)', 'Carbs(g)', 'Fat(g)'])
y_test_df = pd.DataFrame(y_test, columns=['Diet_type'])

# Réinitialiser les index
X_test_df.reset_index(drop=True, inplace=True)
y_test_df.reset_index(drop=True, inplace=True)

# Afficher les exemples correspondant aux indices uniques des prédictions incorrectes
for idx in unique_errors:
    print("Index :", idx)
    print("Caractéristiques :", X_test_df.iloc[idx])  # Accéder aux caractéristiques de l'exemple
    print("Vraie étiquette :", y_test_df.iloc[idx])  # Accéder à la vraie étiquette de l'exemple
    print("Prédiction :", y_pred[idx])  # Afficher la prédiction du modèle
    print("----------------------------------------")
```

## II- Deuxième Modèle utilisé: Modèle de classification basé sur un RandomForest :

1. **Importation des Bibliothèques et des Modules** :
   - Les bibliothèques pandas, scikit-learn sont importées pour la manipulation des données et la construction du modèle.

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

1. **Création du DataFrame à partir des Données** :
   - Les données sont chargées à partir du fichier CSV 'All_Diets.csv' et stockées dans un DataFrame df.
2. **Préparation des Données** :
   - Les caractéristiques (X) et les étiquettes (y) sont définies.
   - Les colonnes numériques et catégorielles sont spécifiées.
3. **Prétraitement des Données** :
   - Des transformateurs sont définis pour normaliser les caractéristiques numériques et encoder les caractéristiques catégorielles à l'aide de ColumnTransformer.
4. **Construction du Pipeline** :
   - Un pipeline est créé, combinant le prétraitement des données et le modèle RandomForestClassifier.
5. **Division des Données** :
   - Les données sont divisées en ensembles d'entraînement et de validation à l'aide de train_test_split.
6. **Entraînement du Modèle** :
   - Le modèle est entraîné sur les données d'entraînement.
7. **Prédictions et Évaluation du Modèle** :
   - Les prédictions sont faites sur l'ensemble de validation.
   - La précision du modèle est calculée à l'aide de accuracy_score et affichée.

```python
# Création du DataFrame à partir des données fournies
df = pd.read_csv('All_Diets.csv')

# Supposons que nous voulons prédire la Cuisine_type
X = df.drop(columns=['Cuisine_type'])
y = df['Cuisine_type']

# Définir les colonnes numériques et catégorielles
numeric_features = ['Protein(g)', 'Carbs(g)', 'Fat(g)']
categorical_features = ['Diet_type', 'Recipe_name', 'Extraction_day', 'Extraction_time']

# Define preprocessing steps for numerical and categorical features
numeric_transformer = StandardScaler()
categorical_transformer = OneHotEncoder(handle_unknown='ignore')

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Define the pipeline with preprocessing and model
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('classifier', RandomForestClassifier())])

# Split data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the model
pipeline.fit(X_train, y_train)

# Predict on validation set
y_pred = pipeline.predict(X_val)

# Evaluate the model
accuracy = accuracy_score(y_val, y_pred)
print(f"Accuracy: {accuracy}")
```

# VII- Validation Croisée

```python
# Création du DataFrame à partir des données fournies
df = pd.read_csv('All_Diets.csv')

# Supposons que nous voulons prédire la Cuisine_type
X = df.drop(columns=['Diet_type'])
y = df['Diet_type']

# Définir les colonnes numériques et catégorielles
numeric_features = ['Protein(g)', 'Carbs(g)', 'Fat(g)']
categorical_features = ['Recipe_name', 'Extraction_day', 'Extraction_time']

# Define preprocessing steps for numerical and categorical features
numeric_transformer = StandardScaler()
categorical_transformer = OneHotEncoder(handle_unknown='ignore')

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Define the pipeline with preprocessing and model
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('classifier', RandomForestClassifier())])

# Evaluate the model using cross-validation
scores = cross_val_score(pipeline, X, y, cv=5)

# Print the mean and standard deviation of the scores
print(f"Mean accuracy: {scores.mean()}")
print(f"Standard deviation: {scores.std()}")
```

- **Dans cette étape on a essayé d'évaluer la performance du modèle de classification utilisant la validation croisée :**

1. **Chargement et Préparation des Données** :
   - Les données sont chargées à partir du fichier CSV 'All_Diets.csv' et stockées dans un DataFrame.
   - Les caractéristiques et les étiquettes sont définies, ainsi que les colonnes numériques et catégorielles.
2. **Prétraitement des Données** :
   - Les caractéristiques numériques sont normalisées et les caractéristiques catégorielles sont encodées à l'aide de ColumnTransformer.
3. **Construction du Pipeline** :
   - Un pipeline est créé, combinant le prétraitement des données et le modèle RandomForestClassifier.
4. **Évaluation du Modèle avec la Validation Croisée** :
   - La méthode cross_val_score est utilisée pour évaluer la performance du modèle en utilisant la validation croisée avec 5 folds.
5. **Affichage des Résultats** :
   - La moyenne et l'écart-type des scores de validation croisée sont imprimés pour évaluer la performance moyenne du modèle et sa variabilité sur les différents folds.

# VIII- Conclusion

**En conclusion, ce projet a suivi une approche méthodique pour construire et évaluer des modèles d'apprentissage automatique pour une tâche de classification. En définissant clairement les objectifs, explorant et préparant les données, et en évaluant plusieurs modèles avec la validation croisée, nous avons pu sélectionner le modèle le plus performant. Cette démarche souligne l'importance d'une méthodologie robuste pour obtenir des résultats fiables et généralisables, avec des implications potentielles pour diverses applications.**